Syed Ameen-ur-Rehman

Description of Distributable Encapsulated Resource Package and Method for Orchestration of Distributed Automation Systems

Master of Science Thesis

Examiners: Dr Niko Siltala, Associate Professor Minna Lanz
Examiners and topic approved by
Faculty council of Faculty of
Automation Engineering
On 8 June 2016

Abstract

Syed Ameen-ur-Rehman: Description of Distributable Encapsulated Resource Package and Method for Orchestration of Distributed Automation System
Tampere University of Technology
Master of Science Thesis, 49 pages, 10 pages.
May 2017
Master's Degree Program in Automation engineering
Major: Factory Automation and Industrial Informatics
Examiners: Dr Niko Siltala, Associate Professor Minna Lanz

Keywords: IEC 61499, Distributed Systems, Encapsulation, Executable Capability, High-Level Control System, Master Recipe, Reconfiguration.

Reconfiguration is critical to meet the current demands of manufacturing and production systems. Rapid reconfiguration can offer small batch sizes, customization and many product variants to the manufacturing companies. This thesis is to implement a novel concept of Resource Description and its Executable Capability.

This is an implementation of theoretical concept by conceiving a use case scenario based on different stakeholders involved in the manufacturing system designing process. The stakeholders are vendors of physical production resource and system integrator who is the consumer of the physical production resource. The scenario is that the vendor provides the control software package along with resource and system integrator uses the software package to develop a control system.

To implement this concept a set of requirements are extracted from resource description and its executable capability concept. Based on these requirements software packages are designed and test case resources are developed. These test case resources are then utilized to design a high-level control system. Realistic test case assembly system is developed and reconfigured.

The obtained results present a realistic view of the concept of resource description in a limited scope. The qualitative analysis of test case scenarios proves the ease which can be achieved in reconfiguration of a manufacturing system if the concept of resource description is used in real-world environments.

Preface

Master of Science Thesis was the first systematic research experience for me and it proved to be an eye opener in my academic career. I am very thankful to Dr Niko Siltala and Dr Eeva Järvenpää for this research opportunity.

I am very grateful to Dr Niko Siltala for providing the help and guidance throughout this thesis work. You gave me very good directions and inspiring instructions, and regularly arranged meetings, which I appreciate a lot. Without your support this thesis would not have been possible. I am also grateful to Dr Niko Siltala for arranging the funding for this thesis. The thesis was partially funded by ReCaM project (Grant agreement No 680759).

I am also thankful to nxtControl Gmbh for software related help. Your help was very beneficial to understand nxtSTUDIO.

Finally, I thank my mom, dad, fiancée and sisters for their unyielding support throughout my academic career.

Tampere 15. May 2017

Syed Ameen-ur-Rehman

Contents

Abstract ii
Preface iii
List of Figures vi
List of Tables vii
List of Abbreviations viii
1. Introduction 1
1.1. Research Questions and Objectives 1
1.2. Research Scope and Limitations 2
1.3. Structure of Thesis 2
2. Theoretical Background 3
2.1. Reconfigurable Manufacturing System 3
2.2. Resource Description and Capability Model 3
2.2.1. Capability Model 4
2.2.2. Resource Description Concept 4
2.2.3. Information Related to this Thesis 6
2.3. IEC 61499 – Standard for programming Distributed Automation Systems . 7
2.3.1. Comparison of IEC 61499 and IEC 61131-3 7
2.3.2. System Model 7
2.3.3. Basic Function Block 9
2.3.4. Composite Function Block 11
2.4. Software Development Environment 11
2.4.1. Composite Automation Type Function Blocks 11
2.4.2. Hardware Controller 12
3. Research Strategy and Methods 13
4. Description of Software Packages 15
4.1. Requirements for Control Software Packages 15
4.2. Developed Solution 16
4.2.1. DERP in Context of IEC 61499 16
4.2.2. DERP in context of CAT type FB 16
4.2.3. First Iteration of the Developed Solution 17

4.2.4. Second Iteration of the Developed Solution 19
4.3. Resources for Test Case 22
4.3.1. Manipulator2DOF DERP 23
4.3.2. Gripper DERP 26
4.3.3. Conveyor DERP 29
4.4. Summary 31
5. Orchestration of Distributed Control System 32
5.1. Orchestration and its Relation to Executable Capability 32
5.2. Master Recipe 33
5.3. High-Level Control System 34
5.3.1. Master Recipe Scheduler 36
5.3.2. Resource Sequence Manager 38
5.4. Test Case Assembly System 40
5.5. Reconfigured Test Case Assembly System 43
6. Results and analysis 45
7. Discussion 47
8. Conclusion and Suggestion 49
References 50
Appendices 52
Appendix A.1: Orchestration Process for Test Case Assembly System 52
Appendix A.2: Orchestration Process for Reconfigured Assembly System 57

List of Figures

Figure 1. Capability Model [3] 4
Figure 2. Resource Model [3] 5
Figure 3. System, Device, Resource and Application Model [6] 8
Figure 4. Graphical Representation of Basic Function Block Type [8] 9
Figure 5. Execution Control Chart [8] 10
Figure 6. Components of DERP (First iteration) 17
Figure 7. Components of DERP (Second Iteration) 19
Figure 8. Interfaces for Parallel Execution of 2ECs 21
Figure 9. Workpiece Models (Developed for simulated system in nxtSTUDIO) 22
Figure 10. Manipulator2DOF Model (Developed in nxtSTUDIO) 24
Figure 11. Interfaces of DERP of Manipulator2DOF (Developed in nxtSTUDIO) .. 25
Figure 12. Internal Components of Manipulator2DOF DERP 25
Figure 13. HMI of Manipulator2DOF (Developed in nxtSTUDIO) 26
Figure 14. Interfaces of DERP of Gripper (Developed in nxtSTUDIO) 27
Figure 15. Internal Components of Gripper DERP 28
Figure 16. HMI of Gripper resource (Developed in nxtSTUDIO) 29
Figure 17. Conveyor Model (Developed in nxtSTUDIO) 29
Figure 18. Interfaces of DERP of Conveyor (Developed in nxtSTUDIO) 30
Figure 19. Internal Components of Conveyor DERP 30
Figure 20. HMI of Conveyor resource (Developed in nxtSTUDIO) 31
Figure 21. Executable Capabilities in context of Orchestration 32
Figure 22. High-Level Control System 35
Figure 23. Interfaces of Master Recipe Scheduler 36
Figure 24. Interfaces of Resource Sequence Manager 38
Figure 25. Layout of Assembly System 40
Figure 26. High-Level Control System of Assembly System 42
Figure 27. Layout of Reconfigured Assembly System 43
Figure 28. High-Level Control System of Reconfigured Assembly System 44

List of Tables

Table 1. Examples of Elements of Resource Model 6
Table 2. Design Science Research Framework 13
Table 3. Test Case Resources, Executable Capabilities and Parameters 22
Table 4. Gripper Models (Developed in nxtSTUDIO) 27
Table 5. Test Case Resources and Executable Capability Commands 31
Table 6. Resource Types and Resource IDs of Assembly System 41
Table 7. Resource Types and Resource IDs of Reconfigured Assembly System 43
Table 8. Use Case Analysis 45

List of Abbreviations

DERP	Distributable Encapsulated Resource Package
HMI	Human Machine Interface
FB	Function Block
CAT	Composite Automation Type
PnP	Pick and Place
ECC	Execution Control Chart

1. Introduction

The products of today need high level of customization and manufacturing speed, due to the highly competitive consumer market. Although the researchers have concluded that the current challenges of legacy production systems can be addressed by distributed automation systems, still the implementation of distributed automation systems open new research areas and pose great many challenges for production and manufacturing systems of tomorrow.

1.1. Research Questions and Objectives

Rapid reconfigurability is of vital importance for reducing product batch size and increasing product customization. Due to the many advantages of rapid reconfigurability, it has become an attractive area of research for manufacturers, consumers and researchers alike. A novel concept of Executable Capabilities is proposed in a Resource Description Concept, which is described in section 2.2. This thesis implements this concept and qualitatively analyse its effect on reconfigurability.

1) What kind of control software packages can be provided by the resource vendors that contributes and assist in development of control system part for reconfigurable manufacturing systems?
2) How Distributed Control System for manufacturing processes can be orchestrated using the vendor provided software packages?

A real-world use case scenario is used to clarify the objectives of this implementation and to introduce the role of different stakeholders whom are to involve if implementation is to become widespread. As stated in question 1, the vendors who manufacture physical production resources should also design and develop software packages. These software packages are to be provided to system integrators along with the physical production resources. The system integrator designs and develops a high-level control system using the vendor provided control software packages. Based on this scenario following are the main objectives of this thesis.

- Develop software packages reflecting physical production resources based on the concept of Resource Description and its Executable Capability.
- How to orchestrate a High-Level Control System using only the exposed part of the developed control software package?

1.2. Research Scope and Limitations

ReCaM is a research project that has been initiated by the European Commission to address the reconfiguration challenges of production and manufacturing systems. The complete project name is *Rapid Reconfiguration of Flexible Production Systems through Capability-Based Adaptation, Autoconfiguration and Integrated tools for Production Planning* (under Grant agreement No: 680759). ReCaM has nine partners and Tampere University of Technology is one of them. The objective of the project is to develop a framework and development tools for reconfigurable plug and produce mechatronic objects. More detailed information about the complete project can be found here [1].

This research comes under the umbrella of Reconfigurable Manufacturing Systems. Although the scope of ReCaM and Reconfiguration is broad in the field of Manufacturing Systems but this research focuses only on the control aspects of the Reconfigurable Manufacturing Systems. This thesis focuses only on the concept of Executable Capability which is developed inside the ReCaM project. The concept of Executable Capability in context of Resource Description is introduced in section 2.2.3.

In addition, following factors are the main limitations for this research work.

- Lack of actual physical production resources.
- Lack of real stakeholders (author acted as Vendor and System Integrator)
- Test case is a model developed by author.
- Although the proposed solution is a general concept but the proposed solution is implemented in a specific development software and hardware (Software is nxtSTUDIO, Hardware is nxtDCSmini)

1.3. Structure of Thesis

This thesis is divided into two main parts. One part is implementation of a theoretical concept and the other is the qualitative analysis of the developed solution. Chapter 2 introduces the theoretical part and all the main concepts which are utilized in this thesis. In addition to the concepts and models, an introduction to the Software and Hardware is also presented in Chapter 2. Chapter 3 introduces the research strategy that is followed during this thesis.

Chapter 4 provide the implemented solution to the theoretical concept. Chapter 5 develops the solution for qualitative analysis.

Chapter 6, 7 and 8 presents analysis of results, evaluation of research and gives future research recommendations.

2. Theoretical Background

This chapter describes the theoretical concepts, which are implemented in this thesis. First, the concept of reconfiguration is introduced in section 2.1. The concept of reconfiguration is used in this thesis to qualitatively analyse the implemented concept. Secondly, the concept of Resource Description and its Executable Capability is described in section 2.2. Thirdly, the basics of IEC 61499 Reference Model are briefly introduced in section 2.3. IEC 61499 is standard for programming distributable automation system, which is used to implement the concept of Executable Capability. Finally, the software development environment and hardware used in this thesis is described in section 2.4.

2.1. Reconfigurable Manufacturing System

A Reconfigurable Manufacturing System is defined in [2] as a system which is designed from the onset to quickly manage structural changes, change in hardware or software component of the system.

The change in system happens at three different levels. These reconfiguration levels are physical reconfiguration, logical reconfiguration and parametric reconfiguration. Physical Reconfiguration means any physical change in the layout of the system. Addition, removal or rearrangement of any physical component of the system would fall in this category of reconfiguration. Logical Reconfiguration means any change in the logical sequence of manufacturing processes without changing system layout. Rerouting and rescheduling of manufacturing processes would fall in this category of reconfiguration. Parametric Reconfiguration means any change in adjustable parameters of the system. An example of parametric reconfiguration would be to change the speed of any system component. [3]

The above-mentioned reconfiguration types are used to demonstrate the reconfigurability of the test case systems, which is described in section 5.5 of this thesis.

2.2. Resource Description and Capability Model

In this section, the main concepts of Resource Description Concept and Capability Model proposed in [3], are described. Example of a simple manipulator is used throughout this section to elaborate Resource Description and Capability Model. Section 2.2.1 contains the introduction of Capability Model and section 2.2.2 contains the introduction of Resource Description Concept. Section 2.2.3 describe the contents of Capability Model and Resource Description Concept in the context of this thesis.

2.2.1. Capability Model

The word ‘Capability’ in the model proposed in [3] is the operational skill which is provided by the physical production resource or which is required by the manufacturing or production process to carry out some task. For example, manipulator can provide the moving capability, which means that the end effector of robot will move from one point in the work space of that manipulator to another point in the same work space. In this model, the capabilities are defined by names and parameters. Because names are not enough to identify the details of capabilities. For example, moving can also mean a work piece moving on the conveyor belt. So, to clearly define the capabilities both names and parameters are used.

The capabilities are further divided into simple and combined capabilities. Combined capabilities are the combination of a set of simple or combined capabilities. For example, Pick and Place is a combined capability which is formed by combining grasping, moving and releasing capabilities. So, these higher level combined capabilities can be achieved by combining low level simple capabilities. Also, these higher level combined capabilities are associated with the lower level simple capabilities with the help of capability associations. And these associations must be satisfied to determine the combined capabilities. The following figure presents the concept of the Capability Model presented in [3].

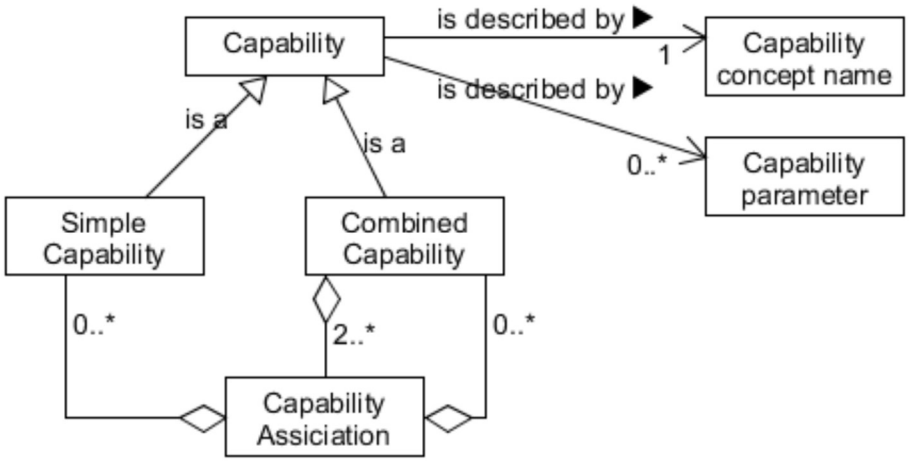

Figure 1. Capability Model [3]

2.2.2. Resource Description Concept

The purpose of the Resource Description Concept is to provide the detailed information regarding the physical production resources for all the stakeholders involved in the manufacturing and production process. Although the information is already available in dispersed form, but this model properly structures the information and provides detailed guidelines on how to divide the information at different levels

of usage. The proposal contains three main document levels namely Abstract Resource Description, Resource Description and Resource Instance Description. [3]

Abstract Resource Description is a document that contains the general information, which is similar in all the physical production resources that will inherit this document. For example, a manipulator can have an Abstract Resource Description that contains all the information that is similar in all the manipulators that will inherit this document. Furthermore, this Abstract Resource Description contains one or more Profiles, which can be implemented to produce physical production resources. The Profiles can be added again and again to increase the functionalities provided by that class of physical production resources.

Resource Description is the file that will be produced by the vendor who is manufacturing the physical production resource. For example, if a vendor 'VA' is producing a two degree of freedom manipulator of type T1 then this vendor will be implementing one of the Profiles provided by the Abstract Resource Description and the vendor specific information will be added to the Resource Description file. Finally, when the physical production resource is ready it is assigned the document named Resource Instance Description, which contains all the information relevant to that specific instance of physical production resource. Resource Instance Description is to travel with the physical production resource throughout its lifetime.

The figure below depicts the relationship between different elements of the Resource Description Concept. [3]

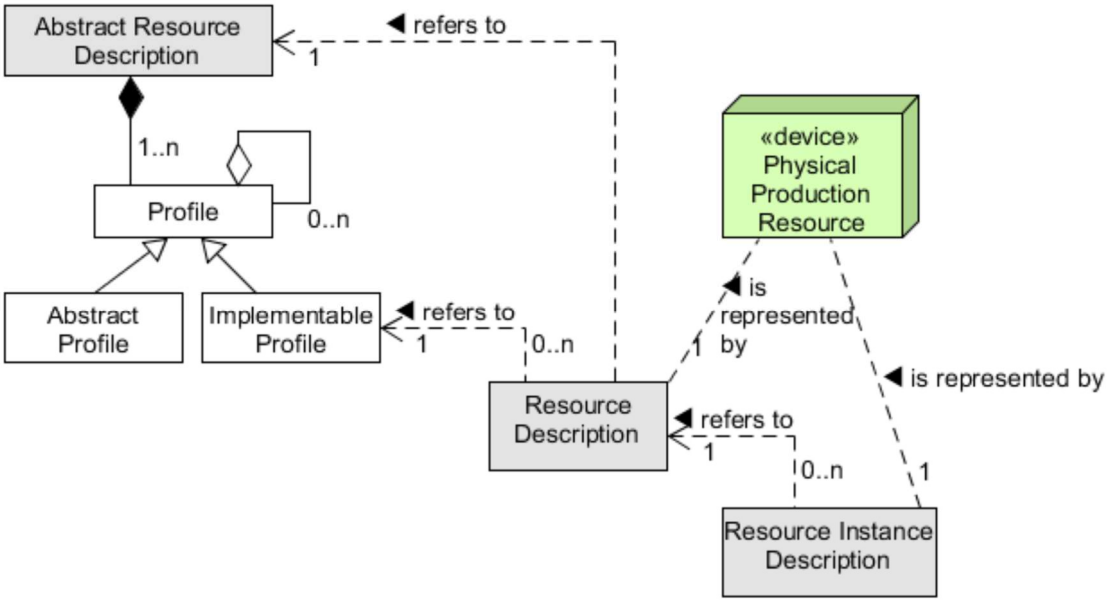

Figure 2. Resource Model [3]

The table below explains the purpose of each document and uses the example of manipulator for understanding of the concept.

Table 1. Examples of Elements of Resource Model

Abstract Resource Description	Similar Kinds of Manipulators, Grippers, Feeders, etc.
Profiles	2 DOF Manipulator, 3 DOF Manipulator, etc.
Resource Description	Vendor Specific 2 DOF Manipulator of specific Type
Resource Instance Description	Vendor Specific 2 DOF Manipulator of specific type and specific serial number

2.2.3. Information Related to this Thesis

The information in the Resource Description Concept and the Capability Model discussed previously is comprehensive, and only the subset of described information is applied is this thesis. Only the control related information is utilised, since this research work is mostly focused on Orchestration of High-Level Control System and Description of the control software packages with which the control system is designed.

In [3] it is stated that Resource Description Concept was further modified to widen the scope of Resource Description from design and reconfiguration planning phase to auto-programming and execution phase of the production system. For this reason, information regarding Capability and Executable Capability was also added to the Resource Description Concept. And this information regarding the Executable Capabilities of physical production resources is the most relevant concept in the context of this thesis. The Executable Capability is defined as follows:

> *"Executable capability is used for controlling the actual execution of the capabilities existing on the resources, i.e. configuring the process parameters and triggering the execution of the capabilities. It differs from simple and combined capability descriptions by having input and output events, which triggers the activities and data inputs and outputs e.g. for setting up the process parameter values. Thus, the executable capabilities need to correspond to the actions that the resources make to complete the task goals (e.g. moveTo, openFingers, closeFingers)." [3]*

The main link between Resource Description Concept and this research is that the control software packages, developed during this research are based on the concept of Executable Capability presented in the Resource Description Concept. The physical production resource, which is mentioned throughout section 2.2, means a mechatronic object such as manipulator, gripper, etc. Also, the concept of resource in Resource Description Concept is different from the concept of resource in the IEC 61499

reference model. The concept of resource in the reference model of IEC 61499 is described in section 2.3.2.

2.3. IEC 61499 – Standard for programming Distributed Automation Systems

IEC 61499 is a reference model that is developed by International Electro-Technical Commission to ease the modelling of Distributed Automation Systems. Since IEC 61499 Reference Model is used in this research to carry out the research objectives presented in section 1.1, so all the relevant concepts of IEC 61499 Reference Model are introduced in this section. Section 2.3.1 describes the important features of IEC 61499 and compare them with the features of IEC 61131-3. In section 2.3.2 modelling concepts of IEC 61499 at system level, device level, resource level and application level are described. Sections 2.3.3 and 2.3.4 introduce the concepts of Basic Function Block and Composite Function Block, which are used extensively in this thesis.

2.3.1. Comparison of IEC 61499 and IEC 61131-3

The main difference between the two standards is that each one is developed with different objectives. The IEC 61499 standard is created with the intent to facilitate distributed automation whereas IEC 61131-3 is developed with the intention to harmonize the programming languages of Programmable Logic Controllers (PLC). In other words, IEC 61499 is a continuation of IEC 61131-3 and it is developed to ease the challenges of reusability, re-configurability, portability and interoperability. [4]

Thomas et al has provided an apt comparison of IEC 61131-3 and IEC 61499 in [5]. Some of the highlights are worth mentioning. IEC 61131-3 supports mainly cyclic execution, whereas IEC 61499 supports event driven execution. In IEC 61131-3 global variables can be used at different levels, whereas IEC 61499 does not allow the use of global variables. In IEC 61131-3 I/O access is achieved via direct access variables, whereas in IEC 61499 I/O access is encapsulated in Service Interface Function Blocks.

All these differences of IEC 61499 and IEC 61131-3 emphasis the intent with which they are conceived. Since this thesis searches for the solutions in scope of distributed automation hence the concepts of IEC 61499 are used throughout this thesis.

2.3.2. System Model

The detailed guidelines, rules and concepts of System Model, Device Model, Resource Model and Application Model, in the context of IEC 61499 Reference Model, are presented in [6] and [7]. Figure 3 is constructed by combining the figures of System Model, Device Model, Resource Model and Application Model, which are presented in [6].

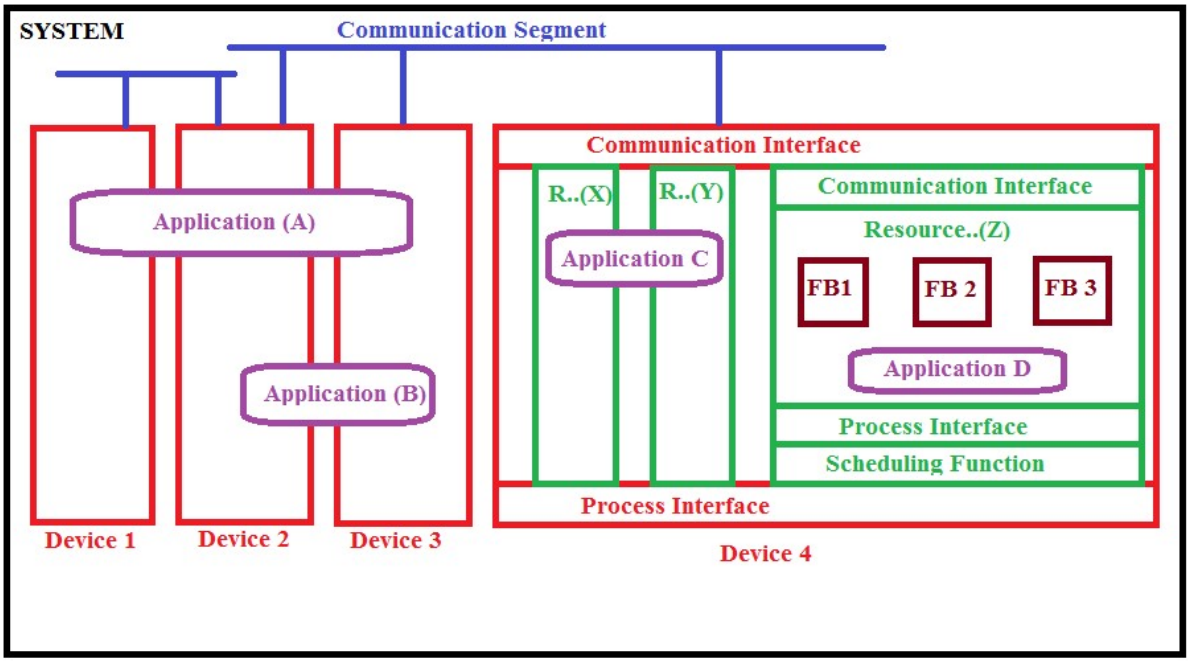

Figure 3. System, Device, Resource and Application Model [6]

In Figure 3 System is composed of one or more communicating devices and it is at the top level of IEC 61499 Reference Model. These Devices communicate with each other via communication segments. A device is control hardware capable of executing the network of function blocks, which are developed using IEC 61499 reference model. The device can communicate both with actual physical modules using process interface and with other devices using communication segments.

A Device is composed of one or more resources. These resources are an abstract division inside the device. Each resource is independent of its execution and is responsible for various tasks. Each resource has a communication interface for higher level communication with other resources on the same devices and remote resources on other devices. Process interface in the resource is responsible for low level I/O communication and Scheduling function is responsible to make sure that all the function blocks inside the resource are executed in the correct order.

Note that the concept of resource in the IEC 61499 Reference Model is different than the concept of Resource Description described in section 2.2. In section 2.2 resource is described as a physical production resource such as manipulator, gripper, etc. The resource in IEC 61499 is further composed of Different Types of Function Blocks, which are described in section 2.3.3 and 2.3.4. For now, a Basic Function Block is the smallest entity in the IEC 61499 Reference Model and it cannot be further divided. In other words, a Basic Function Block is at the lowest level of the IEC 61499 System Hierarchy.

In the IEC 61499 Reference Model an Application is defined as a combination of interconnected network of different types of function blocks. It is also stated that an

application can be distributed among devices and resources. In other words, a single application can run on only one resource or can be distributed among different resources or can be distributed among different devices.

2.3.3. Basic Function Block

The smallest building block of the IEC 61499 Reference Model is a basic type of function block. Basic Function Block is composed of event and data interfaces, Execution Control Chart, Algorithms and internal variables. Figure 4 illustrates the Basic Function Block and its important features.

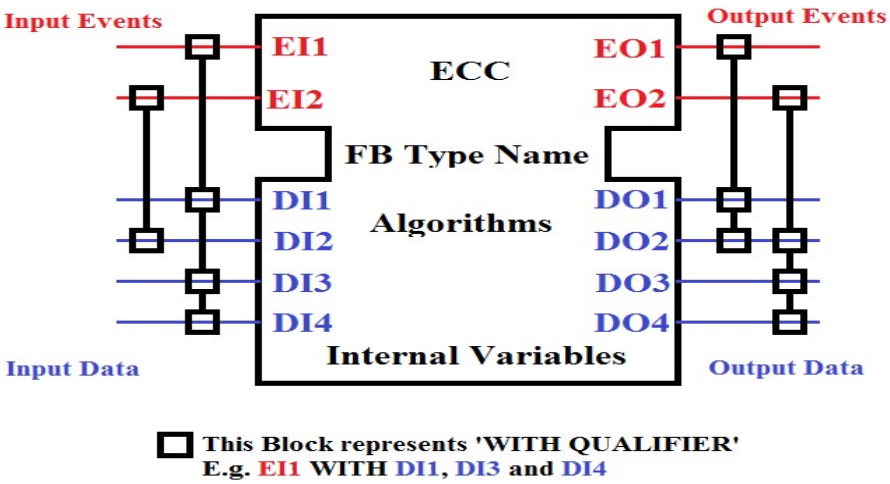

Figure 4. Graphical Representation of Basic Function Block Type [8]

There are two main parts in the definition of a Basic Function Block - the encapsulated part and the exposed part. Encapsulated part is composed of Execution Control Chart (ECC), Algorithms and Internal Variables. Whereas Function Block Type Name, Input Event Interfaces, Output Event Interfaces, Input Data Interfaces and Output Data Interfaces comprises the exposed part of the Basic Function Block.

In Figure 4 Input Event Interfaces are shown at top left and Output Event interfaces are shown at top right. Usually these interfaces are of type EVENT. The IEC 61499 Reference Model also allows the events to be of some other specific type like data types. In this case only similar type of event interfaces can be connected to each other.

The data interfaces are defined with interface name and data type. The Data Types of IEC 61499 and IEC 61131 are the same. In Figure 4 event interfaces are attached to selected data interfaces with the help of square boxes. These square boxes represent the ‘WITH QUALIFIERS’ and they are used to associate data interfaces with event interfaces.

In encapsulated part of the Basic Function Block, Algorithms are implementation specific, which means that they can be written in any PLC programming language,

such as, IEC-61131-3, or any higher-level languages, such as, C, JAVA, etc. The variables can only be defined internally and they can be directly accessed only in the scope of the Function Block within which the variables are defined. They can be accessed by other function blocks only by the means of declared data interfaces of that function block. This property of IEC 61499 eliminates the possibility of global variables and is of utmost importance in the development of Distributed Automated Systems.

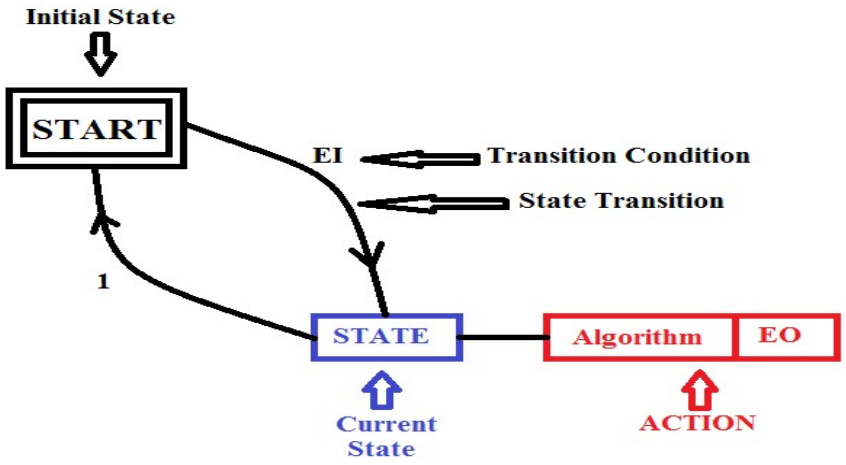

Figure 5. Execution Control Chart [8]

Execution Control Chart (ECC) controls the execution of corresponding function block and is also encapsulated within that function block. The main elements of ECC are shown in Figure 5. In Figure 5 EI means Input Event and EO means Output Event. Every ECC of Basic Function Block must have initial state, which is represented by a double-sided rectangle in Figure 5. When the first event arrives, at any event interface, the execution of function block is initiated from this 'START' state.

The ECC is just like any other state diagram with some slight modifications and it is composed of three main elements in addition to the Start State. These elements are States, State Transitions along with Transition Conditions and State Actions. State Transitions are represented by arrows and the direction of arrow defines the direction of transition. The Transition Conditions are represented by input event and Boolean conditions, which must be met to complete the transition from one state to another state. The Action part of ECC is composed of Algorithm and the output event, which is fired after the execution of that Algorithm is completed.

The Scheduling Function, which is described in section 2.3.2, is part of the resource in which the function blocks are being executed. It is the job of the Scheduling Function to compute the ECC of Basic Function Block, execute the algorithms if required and fire output events if required.

2.3.4. Composite Function Block

As obvious from the name, composite function blocks are composed of basic function block instances and can also contain other composite function block Instances. They are defined in a similar way as the basic function blocks are defined by function block name, input event interface, input data interface, output event interface and output data interface. The only difference is that since the composite function blocks are constructed by interlinking basic function block instances and other composite function block instances so the flow of execution is dependent on the interconnection of function blocks inside the composite function block. Also, the event and data interfaces of composite function blocks are directly linked to the event and data interfaces of function block instances inside the corresponding composite function block.

2.4. Software Development Environment

nxtSTUDIO is a software development environment produced by Austrian based company named nxtControl GmbH. This engineering tool offers the engineering of IEC 61499 based Distributed Automation Systems. nxtSTUDIO offers an integrated solution to a variety of automation tasks such as programming of control logic, distribution of control application among controllers, deployment of control application to controllers, Human Machine Interface (HMI) or Supervisory Control And Data Acquisition (SCADA) based visualization, link to physical production resources, simulation without real installation, etc. More detailed information about nxtSTUDIO can be found in [9].

nxtSTUDIO is selected to implement the proposed solution of this thesis for two main reasons. The first reason is that the software is developed by nxtControl which is one of the partners of the ReCaM project. For this reason, any kind of software related help was easily accessible to the author. The second reason is that the software is based on IEC 61499 Reference Model. Since IEC 61499 Reference Model is used to implement the concept of Executable Capability hence the proposed solution in this thesis could easily be implemented in nxtSTUDIO. For these two reasons, nxtSTUDIO is used throughout this thesis for the implementation of proposed solutions.

2.4.1. Composite Automation Type Function Blocks

The concept of Composite Automation Type (CAT) Function Blocks is developed by nxtControl and CAT concept is based on the Model View Control (MVC) design pattern [4]. CAT combines the control, visualization, field connections, simulation, testing and documentation of a single production resource in one CAT type Function Block [10]. CAT type Function Blocks offer the built-in capabilities to full-fill the requirements of software packages mentioned in section 1.1. For this reason, CAT type

Function Blocks are used to implement the solution proposed to the first research question of this thesis.

2.4.2. Hardware Controller

nxtDCSmini is a controller hardware supplied by nxtControl GmbH. The specifications of nxtDCSmini are available in [11]. Each nxtDCSmini has eight data outputs and each of this data output interface has an adjacent LED, which represents the signal's boolean state. The nxtDCSmini is used in this thesis to test the solutions developed during this research.

3. Research Strategy and Methods

Research is defined as a systematic search of relevant information on a specific topic. Human beings intentionally or unintentionally conduct research in their daily lives such as search of some specific consumer product, service, etc. Researchers intentionally conduct research for different purposes such as finding solution to specific problem, analysing information in specific scope, etc. [12]

An Extended Design Science Research Framework is introduced in [13] for researchers in the field of software engineering. Design Science is a problem-solving research approach, which aims to create new ideas, artefact and concepts via analysis, design and implementation processes. Extended Design Science Research Strategy is chosen for this thesis. The Extended Design Science Research Framework is composed of five phases, which are described as follows:

- **Relevance**: In this phase, the objectives of research are defined along with the criteria of solution and evaluation.
- **The Type of Research**: In this phase, it is determined whether the research is an implementation of a theoretical concept or analysis of an established practice.
- **Theory and Knowledge**: In this phase, the knowledge of theoretical base is researched and reviewed.
- **Design-Build-Test Cycle**: Develop the software artefact by iteratively designing, building and testing each and individual components of the complete solution.
- **Evaluation**: In this phase, the work is evaluated based on established techniques such as use of test case.

The Design Science Research Framework has been utilised in this thesis in the following way.

Table 2. Design Science Research Framework

Design Science Research Framework Phase	**Related Chapter**	**Steps implementing a phase of Design Science Research Framework**
Relevance	Chapter 1	In the Initial thesis meeting, research idea was presented and research questions along with research objectives were discussed.
The Type of Research	Chapter 1	Main objective of this thesis. Implementation of Resource Description Concept and its Executable Capability.
Theory and Knowledge	Chapter 2	Concepts needed for implementation were searched for. Literature review was used to

		figuring out existing concepts, and other relevant, thesis related information
Design-Build-Test Cycle	Chapter 4 and Chapter 5	This phase is conducted in four steps. The requirements are set based on research objectives. The design is proposed based on requirements. The software artefacts are developed based on proposed design. The artefacts are tested against the set requirements. Until the software artefacts meet the requirements the cycle is repeated.
Evaluation	Chapter 6	The implemented solution is qualitatively analysed in context of reconfiguration.

4. Description of Software Packages

This chapter implements the concept of Executable Capability and proposes a solution to the first research question posed in section 1.1 of this document. The requirements of these software packages are introduced in section 4.1 and the proposed solution based on these requirements is presented in section 4.2. Section 4.3 contains the resources for test case and section 4.4 summarizes the complete chapter.

4.1. Requirements for Control Software Packages

The physical production resources, used in the manufacturing and production processes, offer certain capabilities such as robotic manipulator offers move capability. These capabilities also require certain parameters such as destination coordinates in case of robotic manipulator. These capabilities are named here as Executable Capabilities, according the concept of which is described in section 2.2.3.

The basic idea is that the vendors, who manufacture physical production resources, should provide control software packages along with the physical production resources. These control software packages should offer the functionality of physical production resources in the form of Executable Capabilities. The system integrator uses vendor provided control software packages to develop a high-level control system. This high-level control system then controls the physical production system that is composed of those physical production resources.

This thesis focuses mainly on the control software part of the physical production resource. Based on above mentioned scenario following are the requirements that are kept in mind while designing a control software package.

1. The control software package is developed from the perspective of vendors, who manufacture physical production resources. Which means that vendors manufacture the physical production resource along with the control software package.
2. Low-level control logic and Human Machine Interface (HMI) are encapsulated and are vendor specific. Which means same Executable Capability can be implemented differently by different vendors.
3. Executable Capability defines the interface, which are then exposed to system integrator for the development of high-level control system. Which means that high-level control application is developed via only exposed interfaces of vendor provided control software packages.
4. Same physical production resource can execute multiple Executable Capabilities at the same time. If this is the case then vendor must specify this

in resource instance description file. The concept of resource instance description is described in detail in section 2.2.2.

5. Different Executable Capabilities offered by different physical production resources can be executed in parallel.

4.2. Developed Solution

Following the above-mentioned requirements of control software packages, a new term is coined to represent these types of control software packages. The control software package is named as Distributable Encapsulated Resource Package (DERP). The word *distributable* in the term means that control software package is designed for Distributed Automation Systems and *encapsulated* means that the low-level control logic is vendor specific. *Resource package* just mean that this control software package represents an Executable Capability or a set of Executable Capabilities, which are offered by physical production resource as one compilation.

Based on the requirements, following are the four main components of Distributable Encapsulated Resource Package.

1. Low-Level Control Logic.
2. Human Machine Interface.
3. Link to physical production resource.
4. Exposed Interfaces.

4.2.1. DERP in Context of IEC 61499

Distributable Encapsulated Resource Packages are designed following the concepts of IEC 61499 Reference Model. All the relevant concepts and ideas of IEC 61499 Reference Model are presented in section 2.3. There are two important reasons for this selection.

- **Distributable**: First reason is that the IEC 61499 Reference Model is developed from the onset to full fill the requirements of Distributed Automation Systems. DERP is also required to be Distributable that is why IEC 61499 Reference Model is selected.
- **Encapsulated**: Second reason is that the concepts of Function Block and local variables are vital to full fill the encapsulation requirement of DERP.

4.2.2. DERP in context of CAT type FB

The concept of Composite Automation Type (CAT) Function Block (FB) is described in section 2.4.1. CAT type FB is selected to describe and develop DERP for the following reasons.

- CAT combines the concept of Model View Control design pattern in one single Function Block.
- In addition to control, visualization and simulation elements, CAT also offers link to hardware.

The built-in functionalities offered by CAT type FB made it the obvious choice for the development of DERP. Although CAT offered range of functionalities, but the use of CAT and nxtSTUDIO also made the proposed solutions implementation specific. Furthermore, the solution is developed in iterative steps to meet all the requirements, which are set for the solution.

4.2.3. First Iteration of the Developed Solution

This section describes the proposed design of DERP, which is developed in the first iteration. In this iteration, the design of DERP is focused only on the vendor's perspective. The problems with this design and why it is discarded, are described later in this section. The proposed design of DERP in this iteration is depicted in Figure 6.

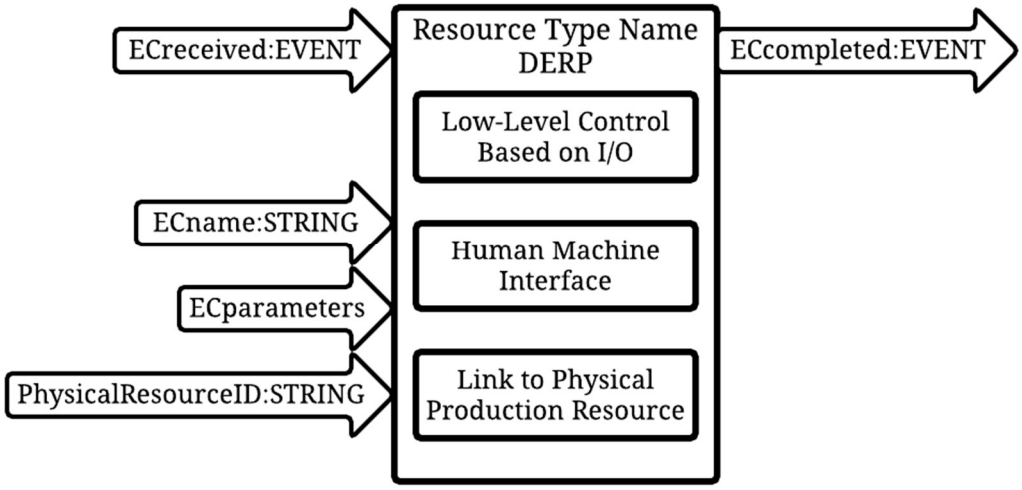

Figure 6. Components of DERP (First Iteration)

The four main components of DERP, which are described in the beginning of section 4.2 are depicted in Figure 6 and are described as follows:

1) **Low-level Control Logic:** This software component of DERP deals with the lower level control of the physical production resource. This is where the Executable Capability along with its parameters is executed. The vendor specificity is most notable in this part. In terms of IEC 61499 Reference Model this part is a basic function block or composite function block. In Figure 6 this component is encapsulated inside DERP.
2) **Human Machine Interface:** HMI is the second internal component and it is integrated into the DERP of physical production resource. The purpose of

integrated HMI is to reduce the system setup time. The idea is that the system integrator develops the HMI of the whole system by combining the HMI components of DERP. In Figure 6 this component is also encapsulated inside DERP.

3) **Link to Physical Resource:** The software component that provides the link to physical production resource is the third internal component of DERP. In terms of IEC 61499 this functionality is provided by the Service Interface Function Blocks. In Figure 6 this component is also encapsulated inside DERP.
4) **Exposed Interfaces:** The exposed interfaces are depicted on the right and left of DERP block in Figure 6. The interfaces on the left brings the information in and interfaces on the right takes the information out of DERP. For example, to perform Executable Capability the information related to that Executable Capability and its parameters is sent using interfaces on the left of DERP block. In terms of IEC 61499 Reference Model the interfaces are described as follows:
    a) **Executable Capability Received (ECreceived):** This is an incoming event type interface. It is attached with ECname and ECparameter data interfaces. The arrival of this event starts the execution of an Executable Capability.
    b) **Executable Capability Completed (ECcompleted):** This is an outgoing event type interface. It is triggered when certain Executable Capability is completed.
    c) **Executable Capability Name (ECname):** This is an incoming data interface, of type STRING. This interface brings the name of the Executable Capability, which is to be executed.
    d) **Executable Capability Parameters (ECparameters):** This represents incoming data interfaces. The data type of these interfaces depends on the type of data, which is brought via these interfaces. The number of these interfaces depends on the definition if Executable Capability.
    e) **Resource Instance ID (PhysicalResourceID):** The purpose of this interface is to uniquely identify the DERP instance. The difference between Resource and Resource instance is described in detail in section 2.2.2.

Although, the design proposed in this iteration meets the requirements, which are described in section 4.1, but this design pose some challenges for system integrator. The first issue with this design is that the instance of DERP reflects an instance of physical production resource. In a high-level control system, the system integrator can only use one instance of DERP for each physical production resource.

Now if the instance of DERP is required to perform same Executable Capability with different parametric values then some other software modules are needed in the high-level control system. Hence, system integrator requires some resource specific software modules which can communicate with the specific instances of DERP whenever required.

The second issue with this design is that the exposed interfaces are resource specific. It means that the interfaces of DERP, based on this design, are different for different resources. This is because the number of parameters, which are required by the Executable Capabilities, might be different for different resources. For example, the Executable Capabilities offered by manipulator resource requires different set of parameters then the Executable Capabilities offered by gripper resource.

The third issue with this design is that the high-level control system, which is created by using only the exposed interfaces of DERP, is going to be application specific. It means that any change is system is going to require a new high-level control system. This makes the reconfiguration of the system time consuming and costly. All these issues are solved in the second iteration by changing the design of DERP.

4.2.4. Second Iteration of the Developed Solution

This section addresses the issues of the first iteration and describes in detail the components of redesigned DERP. The main components of DERP are depicted in Figure 7.

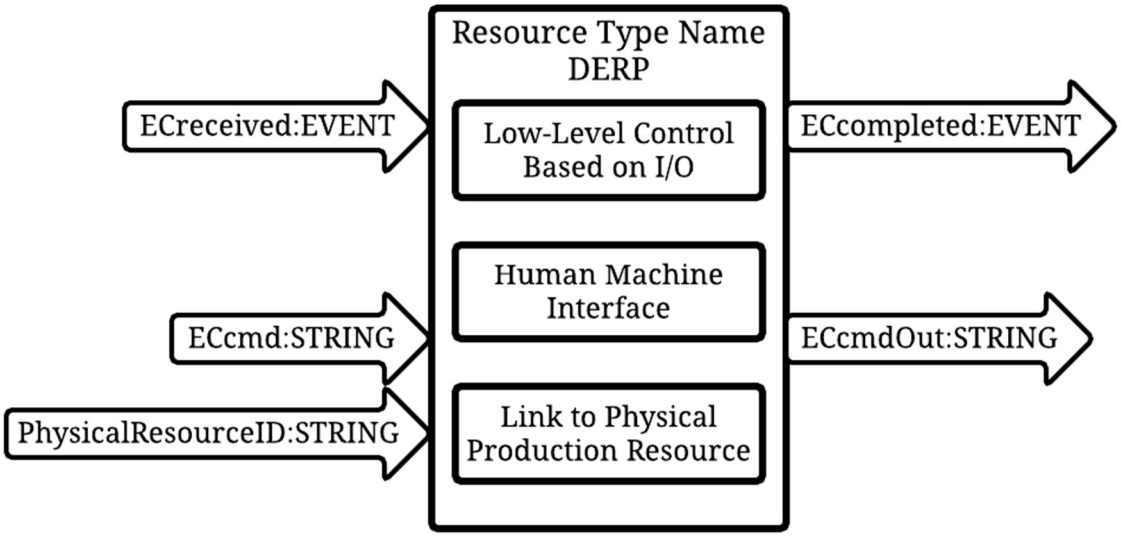

Figure 7. Components of DERP (Second Iteration)

The software components of DERP are described in detail as follows.

1) **Low-level Control Logic:** This software component of DERP deals with the lower level control of the physical production resource. This is where the Executable Capability along with its parameters is executed. The vendor specificity is most notable in this part. In terms of IEC 61499 Reference Model this part is a basic function block or composite function block. In Figure 7 this component is encapsulated inside DERP.

2) **Human Machine Interface:** HMI is the second internal component and it is integrated into the DERP of physical production resource. The purpose of integrated HMI is to reduce the system setup time. The idea is that the System Integrator develops the HMI of the whole system by combining the HMI components of DERP. In Figure 7 this component is also encapsulated inside DERP.
3) **Link to Physical Resource:** The software component that provides the link to physical production resource is the third internal component of DERP. In terms of IEC 61499 this functionality is provided by the Service Interface Function Blocks. In Figure 7 this component is also encapsulated inside DERP.
4) **Exposed Interfaces:** The exposed interfaces are depicted on the right and left of DERP block in Figure 7. The interfaces on the left brings the information in and interfaces on the right takes the information out of DERP. For example, to perform Executable Capability the information related to that Executable Capability and its parameters is sent using interfaces on the left of DERP block. In terms of IEC 61499 Reference Model the interfaces are described as follows.

    a. **Executable Capability Received (ECreceived):** This is an incoming event type interface. It is attached with ECcmd data interface. The arrival of this event starts the execution of an Executable Capability. If a physical production resource offers one Executable Capability at any given time then there is only one ECreceived event interface. This is the case if implemented Executable Capabilities can be triggered only in a sequence and previous one must be always completed before next one can be triggered. In case physical production resource offers multiple Executable Capabilities at the same time then the number of ECreceived event interfaces is equal to the number of Executable Capabilities. For example, if a physical production resource offers Executable Capability A and B at the same time then the number of ECreceived event interfaces is two, one interface for each capability. Similarly, number of ECcmd data interface, ECcompleted event interface and ECcmdOut data interface is also two. Such a scenario is shown in Figure 8.

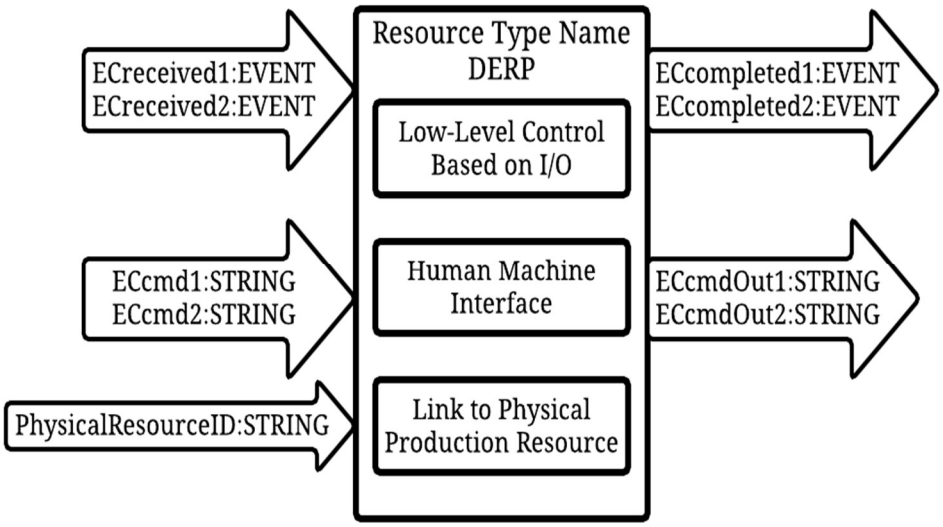

Figure 8. Interfaces for Parallel Execution of 2ECs

b. **Executable Capability Completed (ECcompleted):** This is an outgoing event type interface. It is attached with ECcmdOut data interface. It is triggered when certain Executable Capability is completed.
c. **Executable Capability Command (ECcmd):** This is a data interface of type STRING. It brings the name of the Executable Capability along with its parameters.
d. **Executable Capability Command Out (ECcmdOut):** This is a data interface of type STRING. It is there to provide outgoing information such as status information.
e. **Physical Resource ID:** The purpose of this interface is to uniquely identify the DERP instance. The difference between Resource and Resource instance is described in detail in section 2.2.2.

The design proposed in this section is general purpose and is resource independent. It means that any type of resource can be represented by the interfaces described in this section. All the test case resources, which are developed in this thesis, are based on this design.

4.3. Resources for Test Case

This section contains the description of test case resource packages and test case manufacturing processes, which are developed during this research work. While selecting the test case manufacturing processes following points are kept in mind.

- The manufacturing process must be realistic.
- The manufacturing process is widely used in the real manufacturing environments.
- The manufacturing process is simple and easy to develop.

A simple Pick and Place (PnP) manufacturing process is chosen to be developed because such systems are realistic, easy to develop and are widely used in the manufacturing environment for material handling purposes. The basic resources, that are required for such manufacturing processes, consist of a two degree of freedom manipulator, a finger gripper and a conveyor. Second column of Table 3 contains the list of Executable Capabilities that are deemed fit to full-fill the requirements of the PnP process.

Table 3. Test Case Resources, Executable Capabilities and Parameters

Resource Type	Executable Capability	Parameters
Manipulator 2DOF	MoveAbsolute	[xDest, yDest]
Gripper	Open	
	Close	
	ExternalGrip	
	InternalGrip	
	Release	
	workpiece	[wp1,wp2]
Conveyor	Transport	[xPos, xNeg]
	workpiece	[wp1,wp2,wpi]

Two workpieces are also designed to demonstrate the ExternalGrip and InternalGrip Executable Capabilities of the Gripper resource. Both the workpieces are shown below in Figure 9.

Figure 9. Workpiece Models (Developed for simulated system in nxtSTUDIO)

The square-shaped workpiece is of size 20 pixels x 20 pixels. And U-shaped workpiece is of size (width x height) 60 pixels x 40 pixels. Note that all the visualization

components created during this research work are developed in nxtSTUDIO, that is why all dimensions are in pixels of the visualization environment of nxtSTUDIO.

A simple assembly system is developed using the same resource types that are used in the development of PnP process. The purpose of developing this test case assembly system is to demonstrate the reconfiguration process. The workpieces shown in Figure 9 are assembled as the result of this assembly system.

The role of resource vendor also becomes clear in this chapter. The individual resources and their software components are developed from the perspective of vendor. Master Recipe, which is described in sections 5.1 and 5.2, is created from the perspective of product development team. high-level control system, which is described in section 5.3, is developed from the perspective of system integrator. The above-mentioned test case manufacturing processes and the resource packages are developed using the software development environment nxtSTUDIO, which is introduced in section 2.4.

Following are the three test case resources that are listed in Table 3.

1. Manipulator2DOF
2. Gripper
3. Conveyor

4.3.1. Manipulator2DOF DERP

The Manipulator2DOF resource is a planar robot, which can move a certain object in two degrees of freedom (DOF). The model of Manipulator2DOF resource is depicted in Figure 10. In [14] the workspace of robot manipulator is defined as the set of points that can be reached by its end-effector. The white rectangle in Figure 10 is the workspace of the Manipulator2DOF resource. The width of workspace is 350 pixels and height of workspace is 310 pixels. The tool plate of Manipulator2DOF resource is at top left of workspace. The size of tool plate is 50 pixels x 10 pixels.

The tool plate is attached to the horizontal axis of the Manipulator2DOF and the horizontal axis is attached to the vertical axis of the Manipulator2DOF. The vertical axis is attached to the base of the Manipulator2DOF and the base is shown in Figure 10 at the bottom right of the workspace. The horizontal movement is achieved by moving the tool plate along the horizontal axis and vertical movement is achieved by moving the horizontal axis along the vertical axis. Vertical axis and base are fixed components of Manipulator2DOF.

The MoveAbsolute is the name of the Executable Capability, which is provided by Manipulator2DOF resource. MoveAbsolute means that the Manipulator2DOF resource will move the tool plate to an absolute location inside the workspace of Manipulator2DOF resource. The location of destination coordinates will be provided

as input parameters for MoveAbsolute. These parameters are named as xDest and yDest in Table 3. The range of xDest and yDest is from 0 pixel to 300 pixels.

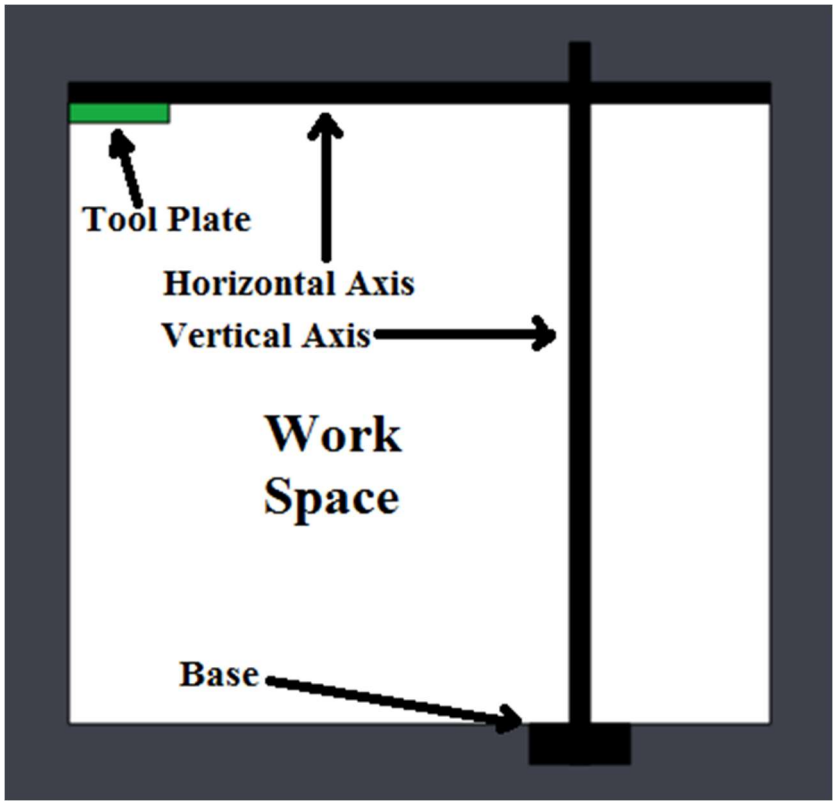

Figure 10. Manipulator2DOF Model (Developed in nxtSTUDIO)

The interfaces of DERP of Manipulator2DOF resource are depicted in Figure 11 and their functional behaviour is the same as described in section 4.2.4. The exposed interfaces of Manipulator2DOF resource has some additional interfaces as well. These interfaces are defined as follows.

1. The INIT and INITO event interfaces in Figure 11 are implementation specific interfaces and these interfaces are used in nxtSTUDIO to initialize some internal Function Blocks of CAT type FB. For example, HMI component of CAT type FB requires initialization.
2. PhysicalGripperID is the data interface, which represents the ID of Gripper resource. The PhysicalGripperID data interface is there for the following two reasons.
    - It represents the instance of Gripper resource that is attached to this instance of Manipulator2DOF resource.
    - To ease the simulation process, which means that the location of Gripper resource is also updated when the location of tool plate of Manipulator2DOF resource is changed.

The ECcmd data interface brings the Executable Capability name and its parameters. In case of Manipulator2DOF the ECcmd has the following format.

$$EC = MoveAbsolute, xDest = (0\ to\ 300), yDest = (0\ to\ 300) \dots \dots ECcmd\ 1$$

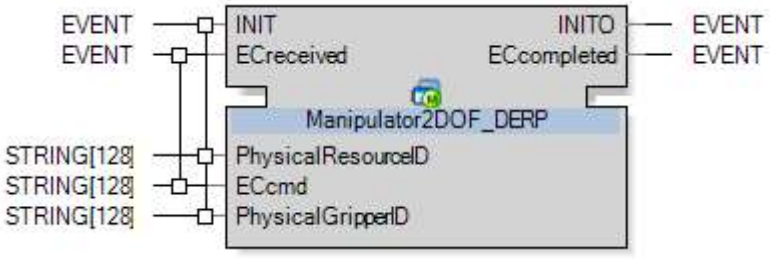

Figure 11. Interfaces of DERP of Manipulator2DOF (Developed in nxtSTUDIO)

The internal components of Manipulator2DOF DERP are the same as described in section 4.2.4. For simulation, two additional components are also added in test case Manipulator2DOF DERP. These components are model of Manipulator2DOF DERP and link to gripper resource. The Link to gripper resource is a transmitter which sends the updated location of Manipulator2DOF tool plate to the attached gripper. The internal components of Manipulator2DOF DERP are shown in Figure 12.

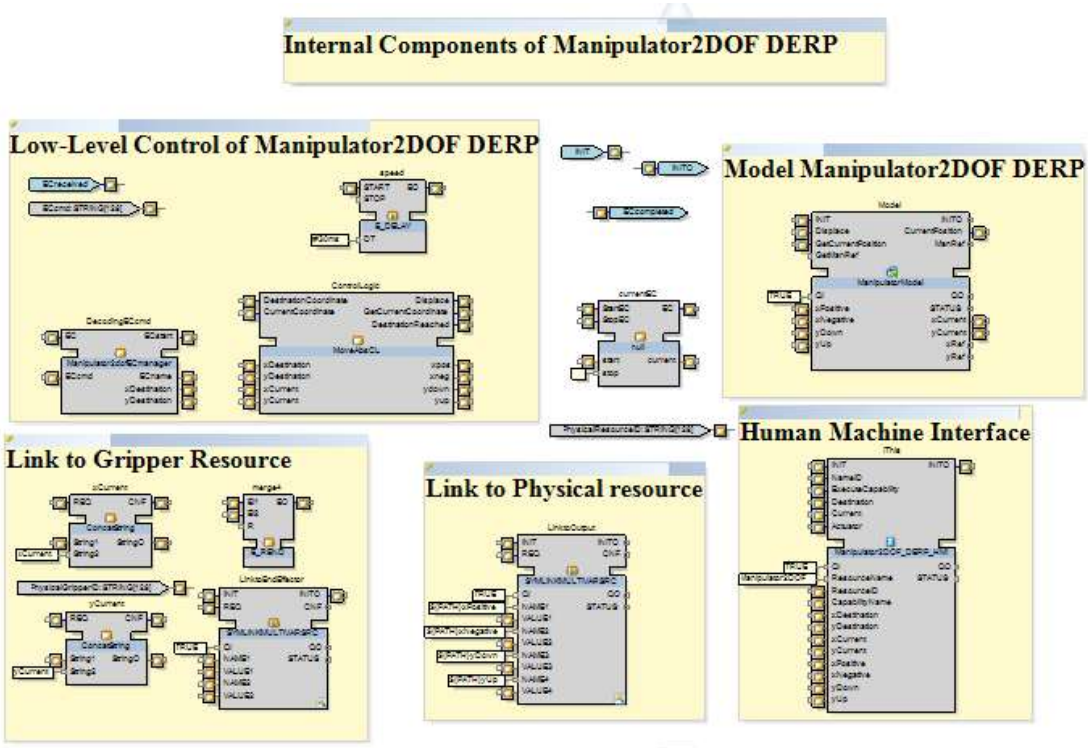

Figure 12. Internal Components of Manipulator2DOF DERP

The low-level control logic of Manipulator2DOF is described as following steps.

1. The low-level control component reads the ECcmd with the arrival of ECreceived event.
2. Low-level control component reads the current location of tool plate of model component.
3. Compare the current location and the destination location.

4. If the xDest is greater than the x coordinate of current location then move the tool plate by one pixel in the positive direction along the horizontal axis.
5. If the xDest is less than the x coordinate of current location then move the tool plate by one pixel in the negative direction along the horizontal axis.
6. If the yDest is greater than the y coordinate of current location then move the tool plate in the downward direction by one pixel along the vertical axis.
7. If the yDest is less than the y coordinate of current location then move the tool plate in the upward direction by one pixel along the vertical axis.
8. Repeat step 2 to 7 until xDest is equal to x coordinate of current location and yDest is equal to the y coordinate of the current location.

Link of the Manipulator2DOF DERP to the Manipulator2DOF physical resource is demonstrated using four digital outputs of the nxtDCSmini10 device on which the instance of Manipulator2DOF DERP is deployed. Two digital outputs are used for demonstrating the movement in horizontal positive and negative direction. The other two digital outputs are used for demonstrating the movement in downward and upward directions.

Note that one instance of Manipulator2DOF provides one MoveAbsolute at any given time. It means that multiple MoveAbsolute cannot be executed at the same time by the same resource.

The HMI of Manipulator2DOF resource is shown below in Figure 13.

Figure 13. HMI of Manipulator2DOF (Developed in nxtSTUDIO)

4.3.2. Gripper DERP

The gripper resource is a finger gripper that can grasp the workpiece internally and externally. The models of Gripper resource are depicted in four different states in Table 4. The gripper is in open state when the fingers are fully extended and the distance between the fingers is 40 pixels. The width of one finger is 5 pixels. The gripper is in close state when the internal distance between the two fingers is 0 pixels. The dimensions of the gripper were chosen keeping in mind the dimensions of Manipulator2DOF and the dimensions of work pieces.

Table 4. Gripper Models (Developed in nxtSTUDIO)

Open	Close	ExternalGrip	InternalGrip
			

The interfaces of DERP of Gripper resource are depicted below in Figure 14. The ECcmd data interface brings the Executable Capability name and its parameters. The list of Executable Capabilities provided by the Gripper resource are presented in Table 3. In case of Gripper the ECcmd has the following format.

$$EC = Open \dots \dots ECcmd\ 2$$

$$EC = Close \dots \dots ECcmd\ 3$$

$$EC = ExternalGrip \dots \dots ECcmd\ 4$$

$$EC = InternalGrip \dots \dots ECcmd\ 5$$

$$EC = Release \dots \dots ECcmd\ 6$$

In addition to the above-mentioned Executable Capability Commands, Gripper DERP also provide the following command for workpiece simulation purposes. The workpiece command is used to make the workpieces visible or invisible inside the Gripper jaws.

$$EC = workpiece, wp1 = (true\ or\ false), wp2 = (true\ or\ false) \dots \dots ECcmd\ 7$$

Figure 14. Interfaces of DERP of Gripper (Developed in nxtSTUDIO)

The internal components of Gripper DERP are the same as described in section 4.2.4. For simulation, two additional components are also added in test case Gripper DERP. These components are model of Gripper DERP and link to Manipulator2DOF resource. The link to Manipulator2DOF resource is a receiver, which receives the updated location of Manipulator2DOF tool plate. The internal components of Gripper DERP are shown in Figure 15.

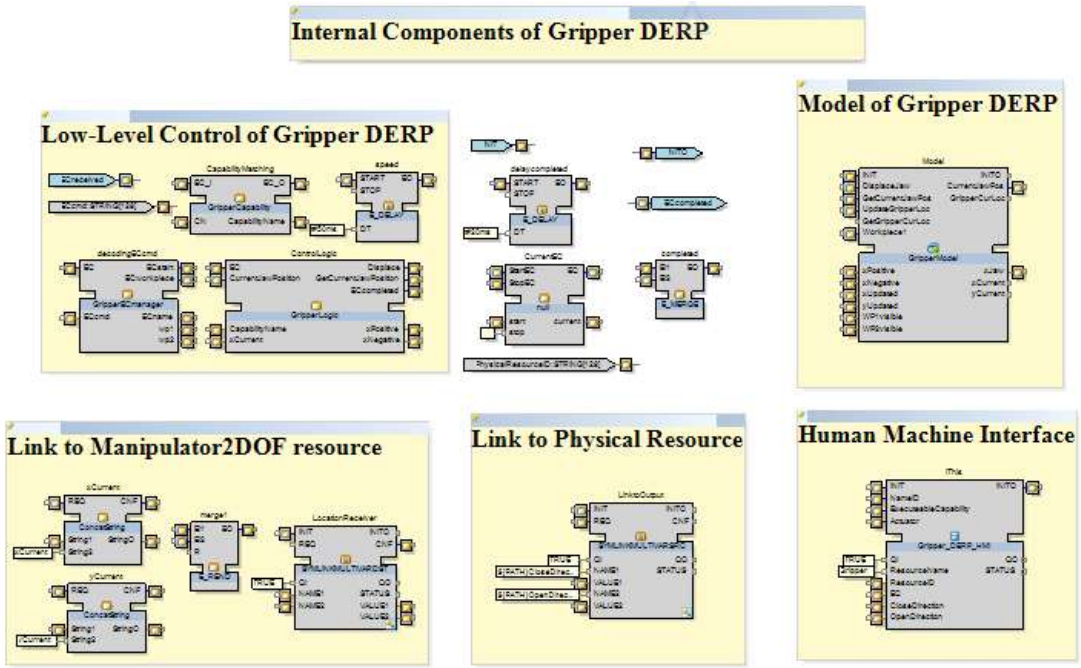

Figure 15. Internal Components of Gripper DERP

The low-level control logic of Gripper is described as following steps.

1. The low-level control component read the name of the executable capability
2. If name is equal to ‘Open’ move the gripper fingers toward the open direction until the gripper fingers are fully extended.
3. If name is equal to ‘Close’ move the gripper fingers toward the close direction until the gripper fingers are fully closed.
4. If name is equal to ‘ExternalGrip’ move the gripper fingers in the close direction and continue to move.
5. If name is equal to ‘InternalGrip’ move the gripper fingers in the open direction and continue to move.
6. If name is equal to ‘Release’ and Gripper fingers are in ExternalGrip state then execute step 2.
7. If name is equal to ‘Release’ and Gripper fingers are in InternalGrip state then execute step 3.

Link of the Gripper DERP to the Gripper physical resource is demonstrated using two digital outputs of the nxtDCSmini10 device on which the instance of Gripper DERP is deployed. One digital output for demonstrating the movement in open direction. The other digital output for demonstrating the movement in close direction.

The HMI of Gripper resource is shown below in Figure 16.

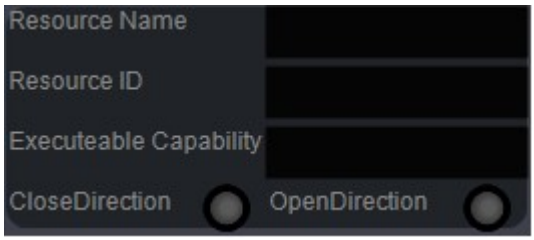

Figure 16. HMI of Gripper resource (Developed in nxtSTUDIO)

4.3.3. Conveyor DERP

The simulation model of Conveyor resource with square-shaped workpiece at left end and U-shaped workpiece at right end is depicted below in Figure 17. The absolute length of the Conveyor surface is 200 pixels.

Figure 17. Conveyor Model (Developed in nxtSTUDIO)

The interfaces of DERP of Conveyor resource are depicted below in Figure 18. The Functional behaviour of Conveyor interfaces is the same as described in section 4.2.4. The ECcmd data interface brings the Executable Capability name and its parameters. In case of Conveyor the ECcmd has the following format.

$$EC = Transport, xPos = (true\ or\ false), xNeg = (true\ or\ false) \dots \dots ECcmd\ 8$$

In addition to the above-mentioned Executable Capability Command, Conveyor DERP also provide the following command for workpiece simulation. The workpiece command is used to make the workpieces visible or invisible at the left most or right most location of the Conveyor. For example, if wp1=true, wp2=false and wpi=0 then workpiece command will make the box workpiece visible at the left most location of the Conveyor model.

$$EC = workpiece, wp1 = (true\ or\ false), wp2 = (true\ or\ false), wpi = (0\ or\ 100) \dots ECcmd\ 9$$

Figure 18. Interfaces of DERP of Conveyor (Developed in nxtSTUDIO)

The internal components of Conveyor DERP are the same as described in section 4.2.4. For simulation, one additional component is also added in test case Conveyor DERP. This component is Model of Conveyor. The internal components of Conveyor DERP are shown in Figure 19.

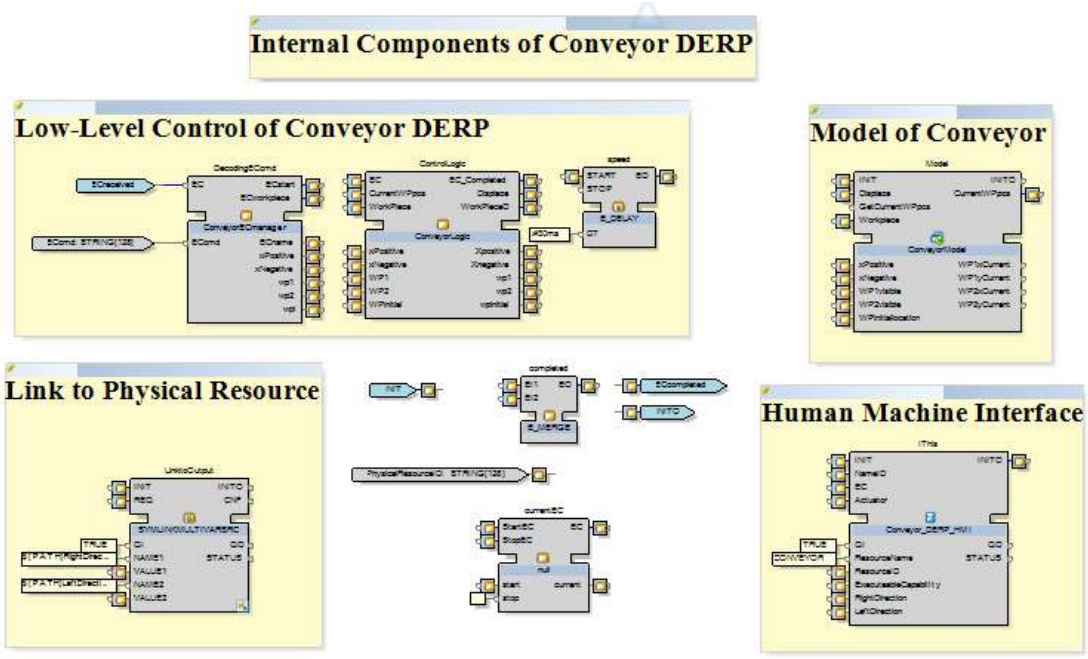

Figure 19. Internal Components of Conveyor DERP

The low-level control logic of Conveyor is described as following steps.

- The low-level control component read the name of the executable capability.
- If xPos is equal to true and xNeg is false then the workpiece is transported from left most location of the conveyor towards the right most location of the conveyor. Which means that the workpiece will move towards forward direction.
- If xPos is equal to false and xNeg is true then the workpiece is transported from right most location of the conveyor towards the left most location of the conveyor. Which means that the workpiece will move towards backward direction.

Link of the Conveyor DERP to the Conveyor physical resource is demonstrated using two digital outputs of the nxtDCSmini10 device on which the instance of Conveyor

DERP is deployed. One digital output for demonstrating the movement in forward direction. The other digital output for demonstrating the movement in backward direction. This demonstrates the controls for an electrical conveyor belt.

The HMI of Conveyor resource is shown below in Figure 20.

Figure 20. HMI of Conveyor resource (Developed in nxtSTUDIO)

4.4. Summary

In this chapter, the concept of Executable Capability is implemented using IEC 61499. Based on proposed design a set of Resources are developed in nxtSTUDIO. These resources provide the list of Executable Capability Commands, which are presented in Table 5. The role of Executable Capabilities is qualitatively analysed in next chapter by utilizing developed resources in a realistic test case.

Table 5. Test Case Resources and Executable Capability Commands

DERP Type	**Executable Capability Command (ECcmd)**
Manipulator2DOF	EC=MoveAbsolute,xDest=(0 to 300),yDest=(0 to 300)
Gripper	EC=Open
	EC=Close
	EC=ExternalGrip
	EC=InternalGrip
	EC=Release
	EC=workpiece,wp1=(true or false),wp2=(true or false)
Conveyor	EC=Transport,xPos=(true or false),xNeg=(true or false)
	EC=workpiece,wp1=(true or false),wp2=(true or false),wpi=(0 or 100)

5. Orchestration of Distributed Control System

In this chapter, a high-level control system is proposed, which is developed from the perspective of system integrator. This high-level control system is developed to demonstrate the use of Executable Capability by system integrator. First, the role of Executable Capability in orchestration process is analysed in section 5.1. Second, a template of master recipe is described in section 5.2. Third, a high-level control system is implemented in section 5.3. Fourth, a test case system is developed in section 5.4, which is based on resources developed in chapter 4. Finally, test case system is reconfigured in section 5.5. The development of test case and its reconfiguration is performed to qualitatively analyse the effect of Executable Capabilities on reconfiguration.

5.1. Orchestration and its Relation to Executable Capability

Orchestration in context of manufacturing processes is defined as discovery, selection and ranking of suitable modules for a given production process [15]. In context of this thesis, module means physical production resource such as Manipulator. Orchestration in manufacturing has a very wide scope. In this thesis, the role of Executable Capabilities in Orchestrating a Distributed Control System is analysed. The concept of Orchestration and its relation to Executable Capabilities proposed in this thesis is depicted below in Figure 21.

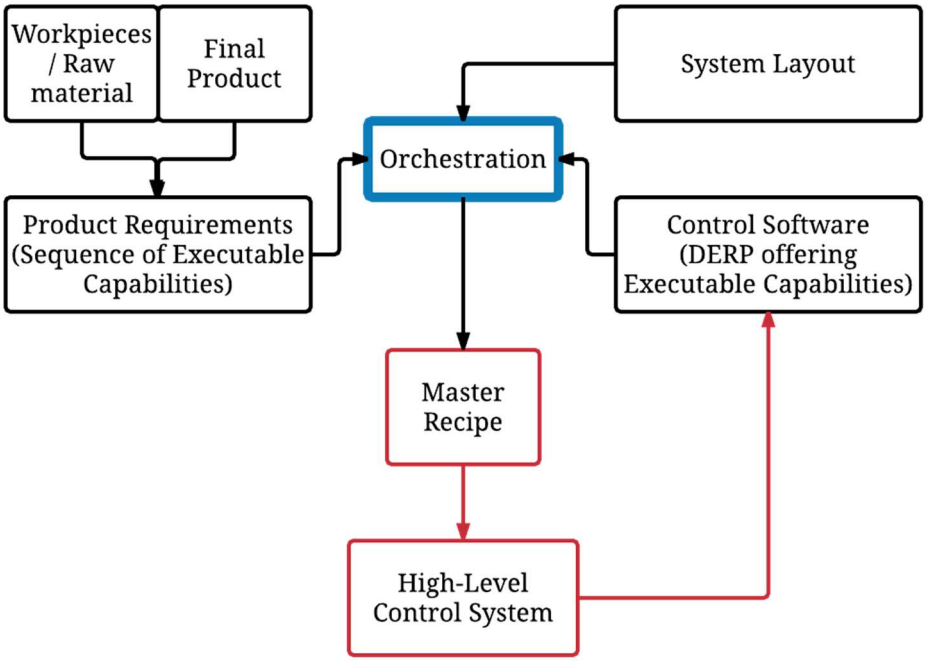

Figure 21. Executable Capabilities in context of Orchestration

In Figure 21 the arrows going towards the Orchestration block represent the flow of information needed to create a Master recipe. Master Recipe is defined in ANSI/ISA 88 as the product recipe which has information related to product, process and equipment [16]. In other words, Master Recipe is composed of information related to product, production equipment and processes of the process cell. In context of this thesis, equipment means physical production resource and processes mean Executable Capabilities. In this thesis, Orchestration is performed manually by performing the following steps.

Step1: Identify the Sequence of Executable Capabilities, which is required to convert the work pieces into final product. Also identify the workpieces or final product for which Executable Capabilities are performed.

Step2: For every sequence required to convert the workpieces into final product, identify the order in which the steps of sequence must be performed. This is done by identifying the previous and next of each step.

Step3: Assign the parameters to each step of the sequence according to the system layout. Match the process requirements of a product to the Executable Capabilities offered by instances of DERP and assign the ResourceIDs to the steps of sequence.

Step4: The output of the Orchestration process is converted into the Master Recipe.

The Master Recipe is then automatically executed by high-level control system. In section 5.2 a template of Master Recipe is provided and all the Master Recipes, for a test case assembly system, are created based on the proposed template. The design of high-level control system, which can automatically execute the Master Recipe, is proposed in Section 5.3.

5.2. Master Recipe

As mentioned in section 5.1, Master Recipe is composed of the information related to product requirements, Executable Capabilities and physical production resources, which offer these Executable Capabilities. In other words, Master recipe is the sequence of Executable Capabilities that are offered by some specific instances of physical production resources and their software counterparts.

In this thesis contents of Master Recipe follows the XML format. A general template of Master Recipe is created and it is given below.

<MasterRecipe>

<Sequence Num=”1”>(Content of Sequence 1)</Sequence>

<Sequence Num=”2”>(Content of Sequence 2)</Sequence>

<Sequence Num=”N”>(Content of Sequence N)</Sequence>

</MasterRecipe>

In the above-mentioned template, N represents total number of steps in the sequence. The contents of each step in the sequence are structured in the following way.

<Sequence
Num="(var1)">Prev=(var2),Next=(var3),ResourceID=(var4),(var5)</Sequence>

The information that can be represented by var1, var2, var3, var4 and var5 is described as follows.

var1: Step number is represented by var1 such as Sequence Num=”5”.

var2: The steps of sequence that just completed before this step. In other words, the steps after which this step should start. Every previous step ends with a hyphen symbol. If this is the first step in the sequence then Prev=0-. If there are more than one previous steps in the sequence then they are represented such as Prev=3-4-.

var3: The step of the sequence that must start after this step. Every next step ends with a hyphen symbol. If this is the last step of the sequence then Next=0-. If there are more than one Next steps then they are represented such as Next=3-4-.

var4: The instance ID of physical production resource is represented by this variable.

var5: This variable represents the actual Executable Capability and its parameters. The format of this part of step in the sequence is described in section 4.3 of this thesis.

5.3. High-Level Control System

As mentioned in section 4.1, a high-level control system is to be developed from the perspective of system integrator. In this thesis, high-level control system is a software system that executes the sequence of Executable Capabilities, which are provided in the Master Recipe. In other words, the main task of high-level control system is to perform the Executable Capabilities according to the schedule provided in the Master Recipe. In context of this thesis schedule means a numbered sequence of Executable Capabilities.

The high-level control system is a high-level control software that is composed of collection of DERPs and some other software artefacts which provide the scheduling functionality. The collection of DERPs provide the Executable Capabilities in the high-level control system. The software artefacts, which provide the scheduling functionality, are developed from the perspective of system integrator to achieve the following objectives of high-level control system.

1. The high-level control system can decode the template of Master Recipe provided in section 5.2.

2. The high-level control system only uses the exposed interfaces of DERPs, which are described in chapter 4.
3. The high-level control system can perform multiple Executable Capabilities in parallel or in series.
4. The high-level control system is easily Reconfigurable.

Since the control software packages proposed in chapter 4 are created using the IEC 61499 Reference Model, hence the high-level control system is also designed following the concepts of IEC 61499 Reference Model. The reasons for this selection are presented in section 4.2.1. Moreover, the proposed design of high-level control system is implemented in nxtSTUDIO.

To accomplish above-mentioned objectives of high-level control system, a special Function Block is proposed. It can read the Master Recipe, decode the Master Recipe and send the Executable Capabilities to the instances of DERP. The Executable Capabilities are sent to instances of DERP according to the sequential information, which is provided in the Master Recipe. This special Function Block is named Master Recipe Scheduler.

The instances of DERP are created from the perspective of vendor as described in chapter 4 and system integrator can only use the exposed interfaces of DERP. So, to add the scheduling functionality to the instances of DERP a special Function Block is proposed in this high-level control system and it is named as Resource Sequence Manager.

The Master Recipe Scheduler, instances of Resource Sequence Manager and instances of DERP are used to design a high-level control system. The block diagram of the proposed high-level control system is shown in Figure 22.

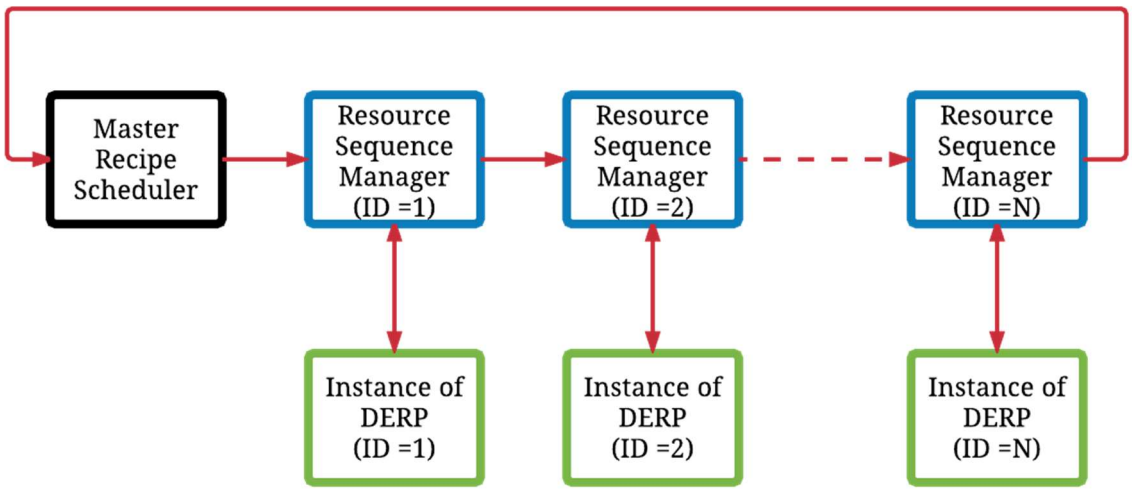

Figure 22. High-Level Control System

In Figure 22 the Master Recipe Scheduler and the Resource Sequence Manager are connected serially in a feedback loop. Each instance DERP instance is assigned an instance of Resource Sequence Manager. The purpose of Resource Sequence Manager is to send the Executable Capability Command to instance of DERP whenever required. Each instance of DERP has a unique instance ID, which is represented as PhysicalResourceID in chapter 4. The instance of Resource Sequence Manager is also assigned the same unique instance ID. In Figure 22 'N' represents the total number of physical production resources of which the manufacturing system is composed of.

In Figure 22 arrows represent the flow of information between Master Recipe Scheduler, Resource Sequence Manager and instances of DERP. This flow of information is composed of event and data interfaces, and this flow of information between Resource Sequence Manager and instance of DERP is bidirectional. The detailed description of this flow of information and functionality of Master Recipe Scheduler and Resource Sequence Manager are presented in sections 5.3.1 and 5.3.2.

5.3.1. Master Recipe Scheduler

The interfaces of the proposed Master Recipe Scheduler are depicted in Figure 23.

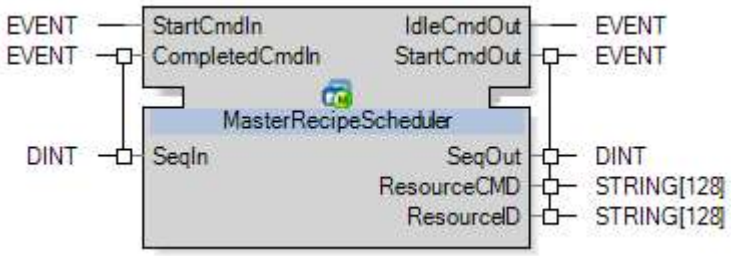

Figure 23. Interfaces of Master Recipe Scheduler

The purpose of output event and data interfaces of Master Recipe Scheduler, which are connected to the first instance of Resource Sequence Manager, are defined as follows. In Figure 22 this is represented by the arrow going from Master Recipe Scheduler towards Resource Sequence Manager (ID=1).

- **IdleCmdOut:** The purpose of this event interface is to check the completion of Executable Capabilities Commands that are being sent before. This event is sent at regular intervals.
- **StartCmdOut:** The purpose of this event interface is to send the step in the sequence whenever required. This event is sent along with SeqOut, ResourceCMD and ResourceID.
- **SeqOut:** The purpose of this data interface, of type INT, is to send the step number of the sequence, which is to be executed.
- **ResourceCMD:** The purpose of this data interface, of type STRING, is to send the Executable Capability Command, which is to be executed. The format of Executable Capability Command is described in section 4.3.

- **ResourceID:** The purpose of this data interface, of type STRING, is to send the resource instance ID of the physical production resource, which is to perform Executable Capability Command.

The purpose of input event and data interfaces of Master Recipe Scheduler, which are connected to the last instance of Resource Sequence Manager are defined as follows. In Figure 22 this is represented by the arrow going from Resource Sequence Manager (ID=N) towards the Master Recipe Scheduler.

- **StartCmdIn:** The purpose of this event interface is to send the next step in the sequence if required.
- **CompletedCmdIn:** The purpose of this event interface is to recognize that some specific step of the sequence in the Master Recipe has been completed. This event is being received along with the SeqIn.
- **SeqIn:** The purpose of this data interface, of type INT, is to receive the step number of the sequence, which has been completed.

The algorithm of Master Recipe Scheduler is designed by keeping track of all the steps in the sequence of Master Recipe. Step Numbers are used to identify steps of sequence in the Master Recipe and they are tracked by using three different states for each step. These states are named Start State, Processing State and Completed State. Start State means that this step is ready to be sent, Processing State means that this step is already sent and Completed State means that this step is Completed. The algorithm is presented as follows.

1. When Master Recipe is received, then decode the Master Recipe and set the states of all the steps in the sequence to Start State.
2. Prepare the list of steps, which are to be sent.
3. Send the step, remove it from the list of steps and set its state to Processing.
4. When ***StartCmdIn*** event is received, then check if there is any other step in the list of steps, which is to be sent.
5. If step found then go to line 3 of algorithm.
6. If no step is found then start the Idle clock by sending ***IdleCmdOut*** events at regular intervals.
7. If ***CompletedCmdIn*** event is received then stop the Idle clock, change the arrived step's state to Completed and check if Master Recipe is completed.
8. If Master Recipe is incomplete then check the next of the arrived sequence and go to line 2 of algorithm.
9. If Master Recipe is completed then wait for the next Master Recipe.

5.3.2. Resource Sequence Manager

The interfaces of the proposed Resource Sequence Manager are depicted in Figure 24. The PhysicalResourceID is an input data interface and it is assigned the same value, which is assigned to its corresponding instance of DERP.

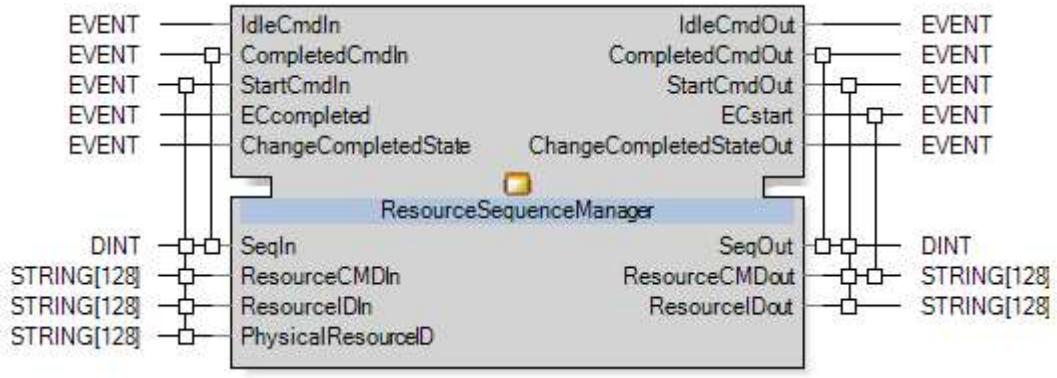

Figure 24. Interfaces of Resource Sequence Manager

The purpose of output event and data interfaces of Resource Sequence Manager, which are connected to next instance of Resource Sequence Manager, are defined as follows. In Figure 22 this is represented by the arrow going from Resource Sequence Manager (ID=1) towards Resource Sequence Manager (ID=2). In Figure 22 if this is the Resource Sequence Manager (ID=N) then only StartCmdOut, CompletedCmdOut and SeqOut are connected to the Master Recipe Scheduler and this is represented by the arrow going from Resource Sequence Manager (ID=N) towards Master Recipe Scheduler.

- **IdleCmdOut:** The purpose of this event interface is to forward the IdleCmdIn event if required.
- **StartCmdOut:** The purpose of this event interface is to forward the step to the next Resource Sequence Manager or to ask the Master Recipe Scheduler for next step in the sequence. This event is sent along with SeqOut, ResourceCMDout and ResourceIDout.
- **CompletedCmdOut:** The purpose of this event interface is to forward CompletedCmdIn event or to inform the Master Recipe scheduler that some specific step in the sequence is being completed by this Resource Sequence Manager. This event is sent along with SeqOut.
- **SeqOut:** This is same as described in section 5.3.1.
- **ResourceCMDout:** This is same as ResourceCMD in section 5.3.1.
- **ResourceIDout:** This is same as ResourceID in section 5.3.1.

The purpose of input event and data interfaces of Resource Sequence Manager, which are connected to previous instance of Resource Sequence Manager, are defined as follows. In Figure 22 this is represented by the arrow going from Resource Sequence

Manager (ID=1) towards Resource Sequence Manager (ID=2). In Figure 22 if this is the Resource Sequence Manager (ID=1) then event named CompletedCmdIn is not connected to Master Recipe Sceduler.

- **IdleCmdIn:** The purpose of this event interface is to check if a step in the sequence is completed.
- **StartCmdIn:** The purpose of this event interface is to perform the step in the sequence if required.
- **CompletedCmdIn:** The purpose of this event interface is to inform the Master Recipe Scheduler that some specific step in the sequence is completed. This event is received along with the SeqIn.
- **SeqIn:** This is same as SeqIn in section 5.3.1.
- **ResourceCMDIn:** This is same as ResourceCMD in section 5.3.1.
- **ResourceIDIn:** This is same as ResourceID in section 5.3.1.

The purpose of input and output interfaces of Resource Sequence Manager, which are connected to corresponding instance of DERP, are defined as follows. In Figure 22 this is represented by the bidirectional arrows between Resource Sequence Managers and DERPs.

- **ECstart:** The purpose of this event interface is to send the Executable Capability Command to the DERP.
- **ECcompleted:** The purpose of this event is to inform the Resource Sequence Manager that the requested Executable Capability Command is completed. This event is sent along with ResourceCMDout.
- **ResourceCMDout:** This is same as ResourceCMD in section 5.3.1.

The algorithm of Resource Sequence Manager is designed by keeping track of the steps in the sequences, which Resource Sequence Manager can send to corresponding DERP. Step numbers and ResourceIDs are used to identify concerned steps in the sequence and they are tracked by using three different states for each step of the sequence. These states are named as Idle State, Processing State and Completed State. Idle State means that the corresponding instance of DERP is idle, Processing State means that some specific step in the sequence is sent to corresponding instance of DERP and Completed State means that corresponding instance of DERP has completed some specific step in the sequence. Additionally, two internal variables, namely step variable and state variable are used. The algorithm is presented as follows.

1. Initially step variable is equal to 0 and state variable is equal to Idle.
2. Wait for ***StartCmdIn*** event or ***IdleCmdIn*** event or ***CompletedCmdIn*** event or ***ECcompleted*** event.
3. When ***StartCmdIn*** event is received then set ***SeqOut*** equal to ***SeqIn***, set ***ResourceCMDout*** equal to ***ResourceCMDIn***, set ***ResourceIDout*** equal to ***ResourceIDIn*** and check if the ***ResourceID*** is equal to ***PhysicalResourceID***.

4. If ***ResourceID*** is equal to ***PhysicalResourceID*** then change the step variable to ***SeqIn***, state variable to Processing, send the ***ECstart*** event to corresponding DERP and trigger the ***StartCmdOut*** event.
5. If ***ResourceID*** is not equal to ***PhysicalResourceID*** then trigger the ***StartCmdOut*** event.
6. Go to line 2 of algorithm.
7. When ***CompletedCmdIn*** event is received then set ***SeqOut*** equal to ***SeqIn*** and trigger a ***CompletedCmdOut*** event.
8. Go to line 2 of algorithm.
9. When ***IdleCmdIn*** event is received then check the state variable.
10. If state variable is equal to Idle or is equal to Processing then trigger ***IdleCmdOut*** event.
11. If state variable is equal to Completed then set ***SeqOut*** equals to step variable and trigger ***CompletedCmdOut*** event.
12. Go to line 1 of algorithm.
13. When ***ECcompleted*** event is received then set the state variable to Completed.
14. Go to line 2 of algorithm.

5.4. Test Case Assembly System

In this section description of a test case assembly system is presented. This assembly system is developed using the test case resources, which are described in section 4.3. The virtual model of the physical layout of the assembly system is depicted in Figure 25. The resources for the assembly system are being assigned ResourceIDs and they are presented in Table 6. The resources are labelled in the Figure 25 with ResourceIDs. The assembly system is composed of two instances of Manipulator2DOF type resource, two instances of Gripper type resource and four instances of Conveyor type resource.

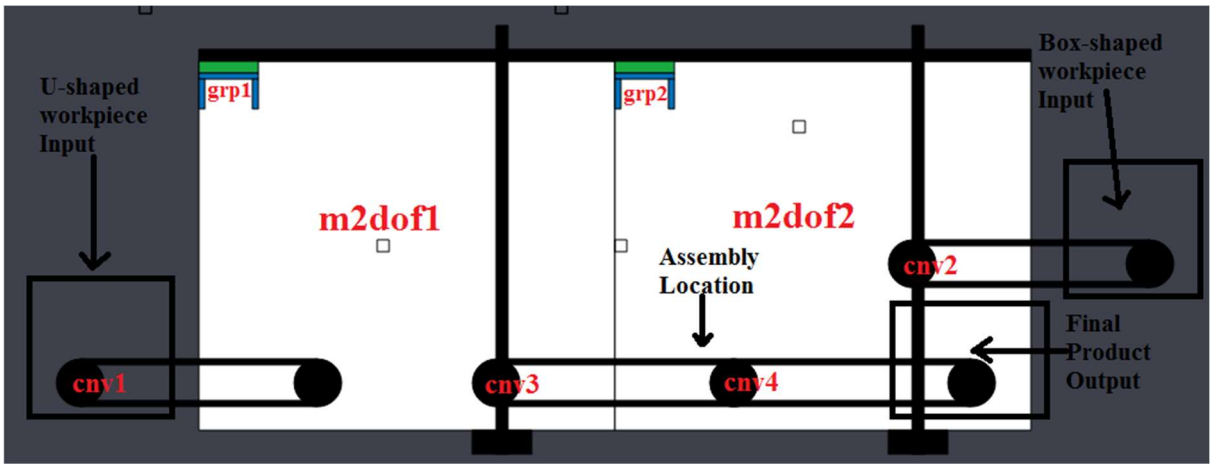

Figure 25. Layout of Assembly System

The Gripper type resource with ResourceID equals to *grp1* is attached to Manipulator2DOF type resource with ResourceID equals to *m2dof1*. Similarly, Gripper type resource with ResourceID equals to *grp2* is attached to Manipulator2DOF type resource with ResourceID equals to *m2dof2*.

The Conveyor type resource with ResourceID equals to *cnv1* and *cnv2* acts as input feeders to the assembly system. Which means that *cnv1* and *cnv2* are used to feed the work pieces into the system. These work pieces are described in section 4.3. The Conveyor type resource with ResourceID equals to *cnv3* acts as an assembly conveyor. Which means that the work pieces are assembled on *cnv3*. The Conveyor type resource with ResourceID equals to *cnv4* acts as an output feeder. Which means that cnv4 takes the assembled product out of the system.

Table 6. Resource Types and Resource IDs of Assembly System

Resource Type	Manipulator 2DOF	Gripper	Conveyor
Resource IDs	m2dof1	grp1	cnv1
	m2dof2	grp2	cnv2
			cnv3
			cnv4

The assembly process is described as follows.

- Process 1: U-shaped workpieces enter the system from the left of *cnv1* and the work piece is transported to pick-up location at the right most of *cnv1*.
- Process 2: Box-shaped workpieces enter the system from the right of *cnv2* and is transported to pick-up location at the left most of *cnv2*.
- Process 3: The end-effectors of *m2dof1* is moved to location right above the pick-up location of U-shaped workpiece. The fingers of *grp1* are moved to close position. The end-effector is moved to pick-up location.
- Process 4: The *grp1* internally grip the U-shaped workpiece. The end-effector of *m2dof1* is moved to location right above the pick-up location of U-shaped workpiece. The end-effector is moved to location right above the drop-off location, which is left most point of *cnv3*. The end-effector is moved to the drop-off location.
- Process 5: The U-shaped workpiece is moved to assembly location which is right most point of *cnv3*.
- Process 6: The end-effector of *m2dof2* is moved to location right above the pick-up location of Box-shaped workpiece. The fingers of *grp2* are moved to open position. The end-effector is moved to pick-up location.
- Process 7: The end-effectors of *m2dof2* is moved to location right above the pick-up location of box-shaped workpiece. The end-effector is moved to location right above the assembly location. The end-effector is moved to the assembly location.

- Process 8: The *grp2* release the box-shaped workpiece at the assembly location. The final product is assembled. The end-effector of *m2dof2* is moved to location right above the assembly location.
- Process 9: The final product is transported out of the system via *cnv4*.

The Master Recipe for this test case assembly system is created by following the steps, which are described in section 5.1. The complete process of creating the Master Recipe of this assembly system is attached in the Appendix A.1 of this thesis.

The high-level control system for this test case assembly system is developed using the design proposed in section 5.3. The development environment in which this high-level control system is created is nxtSTUDIO.

Each instance of physical production resource used in this assembly system is represented by an instance of DERP of that specific resource type in the high-level control system. For example, the Manipulator2DOF type resource with ResourceID equals to *m2dof1* is represented by instance of Manipulator2DOF_DERP in the high-level control system and is given the PhysicalResourceID equals to *m2dof1*. Since the assembly system is composed of eight instances of physical production resources hence the high-level control system is also composed of eight instances of DERP. Each instance of DERP is assigned an instance of Resource Sequence Manager and the same ResourceID is assigned to corresponding Resource Sequence Manager.

The block diagram of high-level control system for this test case assembly system is depicted in Figure 26. Note that this high-level control system is independent of the sequence in which the instances of DERP are physically connected. For example, Manipulator2DOF_DERP with ResourceID equals to *m2dof1* can be represented by any of the DERPs presented in Figure 26. But if DERP1 is represented by Manipulator2DOF_DERP with ResourceID equals to *m2dof1* then RSM1 should also have the ResourceID equals to *m2dof1*. Moreover, the ResourceIDs must be unique. Which means that there should be only one resource type with the ResourceID equals to *m2dof1*.

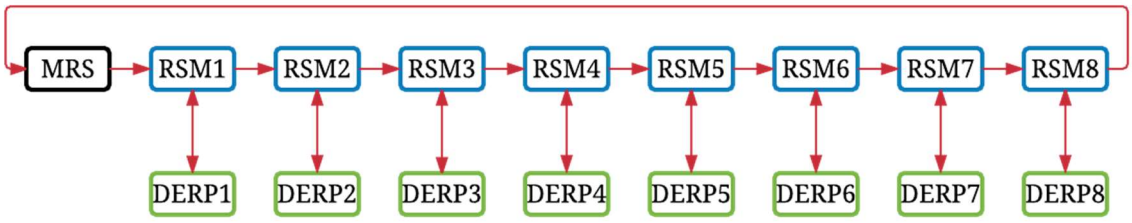

Figure 26. High-Level Control System of Assembly System

5.5. Reconfigured Test Case Assembly System

The definition of logical, parametric and physical reconfiguration is provided in section 2.1. In context of this thesis, logical reconfiguration means to change the sequences of Executable Capabilities in the Master Recipe, parametric reconfiguration means to change the parameters of Executable Capabilities in the Master Recipe and physical reconfiguration means to rearrange, add or remove the physical production resources of the assembly system.

In this section, the assembly system is reconfigured so that the production capacity of the assembly system is doubled. This is achieved by adding two instances of Conveyor type resources to the assembly system described in section 5.4 and rearranging the layout. The layout of this reconfigured assembly system is depicted in Figure 27.

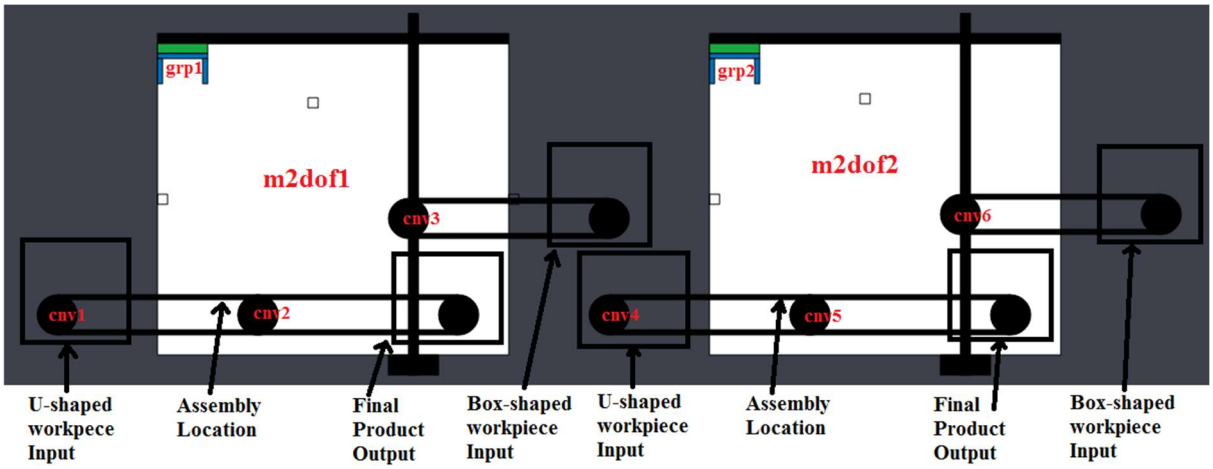

Figure 27. Layout of Reconfigured Assembly System

This reconfigured assembly system is composed of two instances of Manipulator2DOF type resource, two instances of Gripper type resource and six instances of Conveyor type resource. All the instances of resource with their corresponding ResourceIDs are presented in Table 7.

Table 7. Resource Types and Resource IDs of Reconfigured Assembly System

Resource Type	Manipulator 2DOF	Gripper	Conveyor
Resource IDs	m2dof1	grp1	cnv1
	m2dof2	grp2	cnv2
			cnv3
			cnv4
			cnv5
			cnv6

The Gripper type resources are attached to Manipulator2DOF type resource the same way as they are attached in section 5.4. The Conveyor type resource with ResourceID equals to *cnv1*, *cnv3*, *cnv4* and *cnv6* acts as input feeders to this reconfigured assembly

system. The Conveyor type resource with ResourceID equals to *cnv1* and *cnv4* acts as an assembly conveyor. The Conveyor type resource with ResourceID equals to *cnv2* and *cnv5* acts as an output feeder.

The assembly process is described as follows.

- Process 1: U-shaped workpieces enter the system from the left of *cnv1* and *cnv4*, and the work pieces are transported to assembly location at the right most of *cnv1* and *cnv4*.
- Process 2: Box-shaped workpieces enter the system from the right of *cnv3* and *cnv6*, and are transported to pick-up location at the left most of *cnv3* and *cnv6*.
- Process 3: The end-effectors of *m2dof1* and *m2dof2* is moved to locations right above the pick-up location of box-shaped workpieces. The end-effectors are moved to pick-up location.
- Process 4: The *grp1* and *grp2* externally grip the box-shaped workpieces.
- Process 5: The end-effectors of *m2dof1* and *m2dof2* are moved to location right above the pick-up location of box-shaped workpieces. The end-effectors are moved to location right above the assembly location. The end-effectors are moved to the assembly location.
- Process 6: The *grp1* and *grp2* release the box-shaped workpieces at the assembly location. The final product is assembled. The end-effectors of *m2dof1* and *m2dof2* are moved to location right above the assembly location.
- Process 7: The final products are transported out of the system via *cnv2* and *cnv5*.

The Master Recipe for this reconfigured assembly system is created the same way as it is created in section 5.4. The complete process of creating the Master Recipe of this reconfigured assembly system is attached in the Appendix A.2 of this thesis. And the high-level control system for this reconfigured assembly system is developed the same way as it is developed in section 5.4. The only difference is that the number of instances of DERP are increased to 10 in high-level control system of this reconfigured assembly system. The block diagram of high-level control system for this reconfigured assembly system is depicted below.

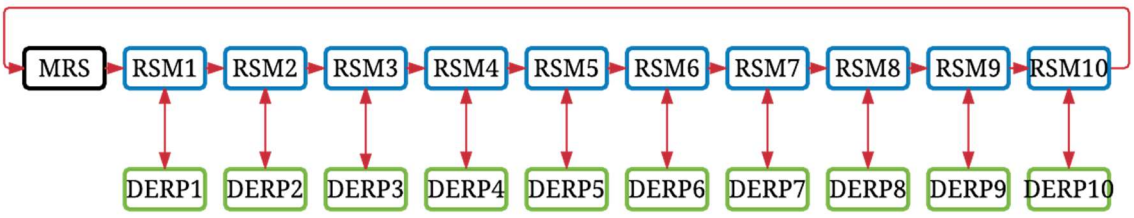

Figure 28. High-Level Control System of Reconfigured Assembly System

6. Results and analysis

In this thesis, a theoretical concept of Resource Description and its Executable Capabilities is implemented. The implementation is performed in two steps. In first step, the control software package is designed based on the requirements which are extracted from Resource Description Concept. In step 2, the high-level control system is designed by using the exposed interfaces of control software package which is designed in step 1.

The implementation is tested by developing a realistic test case system based on the designs proposed in step 1 and step 2. The test case system is developed in three phases. In first phase, resources for the test case are developed. These resources can meet all the requirements which are set in step 1. In second phase, a realistic manufacturing system is developed by combining the resources of phase 1. Finally, in phase three the system developed in phase two is reconfigured for qualitative analysis.

The effect of the proposed designs, which are described in chapter 4 and 5, on the reconfiguration is qualitatively analysed using use case scenarios. The use cases are based on the changes to test case system developed in section 5.4 and resource IDs are used to identify the instances of DERP. The effect of reconfiguration is described on master recipe, high-level control system and instances of DERP. The use case analysis is presented in Table 8.

In Table 8 use case number 5 is demonstrated in section 5.5 by reconfiguring the test case assembly system presented in section 5.4. The implemented solution of this thesis does not require any change in the high-level control system if the system undergoes through parametric, logical or physical (rearrangement of resources) reconfiguration. Even in use case 4 the high-level control system can still work without any changes. If no changes are made in the high-level control system in use case 4 then there will be some instances of DERP which will be needless and could be removed.

The high-level control system requires mandatory change only in use case 5 when new resources are added to the system. Even in this case the time it takes to add new instances of DERP in the proposed high-level control system is less than 3 minutes.

Table 8. Use Case Analysis

No	Use Case	Change Type	Master recipe	Instances of DERP	High-Level Control System
1	The pick and place path of	Parametric reconfiguration.	Yes, the parametric content of	No new instance of	No change.

	m2dof1 is changed.		master recipe requires change.	DERP is required. No change in any instance of current DERP.	
2	The input feeders are changed, which means that box-shaped workpiece enters from cnv1 and U-shaped workpiece enters from cnv2.	Logical reconfiguration.	Yes, the sequence of Executable Capability Commands in master recipe requires change.	No new instance of DERP is required. No change in any instance of current DERP.	No change.
3	The location of gripper resources is changed, which means that grp1 is now attached to m2dof2 and grp2 is now attached to m2dof1.	Rearrangement of resources. Physical reconfiguration.	Yes, the Resource IDs of Executable Capability Commands in master recipe requires change.	No new instance of DERP is required. No change in any instance of current DERP.	No change.
4	The input feeder is now attached to cnv3 which means m2dof1, grp1 and cnv1 are removed from the assembly process.	Removal of resources. Physical reconfiguration.	Yes, the content of master recipe requires change.	No new instance of DERP is required. Some instances of current DERP are removed.	Yes, some instances of DERP are removed.
5	The production capacity is changed by adding more resources. The reconfigured test case described in section 5.5.	Addition of resources. Physical reconfiguration.	Yes, the content of master recipe requires change.	Yes, new instances of DERP are required.	Yes, some instances of DERP are added.

7. Discussion

This implementation adds a practical dimension to the resource description and its executable capability concept. This thesis demonstrates the role of different stakeholders involved in developing a manufacturing system. This thesis also analyses the effect of resource description concept in context of reconfiguration.

The research is implementation specific. The use of IEC 61499 and nxtSTUDIO in design and development process have restricted the scope of research in a constrained environment. All the software artefacts are developed and tested in nxtSTUDIO which makes the implementation of this thesis prone to interoperability issues. This means that the results obtained in this thesis are fruitful only if the vendors develop the control software packages using nxtSTUDIO, product development team follows the template provided in the thesis and system integrator uses nxtSTUDIO as well.

In this thesis, a realistic manufacturing system is conceived for the test case but still it is fairly simple for a real-world manufacturing system. The real-world systems are of much higher complexity and resources used are numerous. Which means that the implementation might face some other challenges when the system is composed of let say hundreds of resources and master recipe is composed of thousands of sequences.

The model elements of the test case resources were useful in creating visual simulation of test case systems and they were significant in demonstrating reconfiguration. But the use of model elements also restricted the results of implementation to virtual world. The developed implementation was deployed and distributed on a set of physical controllers. And the results were demonstrated only on the digital outputs of physical controllers. This lack of physical production resources also limited the scope of implementation. It means that if the proposed implementation is conducted using physical production resource then the use of actual resources might add some other challenges. However, from deployment and controller utilization perspective the approach is valid.

The implemented elements of this thesis are developed from the perspective of different stakeholders, such as vendor of physical production resources, system integrator who is the user of physical production resources, product development team who is the creator of master recipe. In real-world, each of these stakeholders might have a different perspective towards manufacturing systems. Lack of the input from real stakeholders also narrowed the perspective with which the implementation is performed. It means that if real stakeholders were involved then the requirements set for each implementation phase might change.

Following five approaches could have added significant authenticity to the research.

First approach: The use of cross platform environment for the development of implemented proposals. As mentioned earlier, all the software artefacts were developed in the same environment. Different stakeholders use different platforms so it would have significantly widened the scope of research if the test case resources as well as test case systems were developed on different development environments.

Second approach: The complexity of test case resources and test case systems should have increased to match at least the complexity of small sized manufacturing systems of real-world. This approach could have added the authenticity to the proposed implementation.

Third approach: Significant time of this thesis was spent on developing the model elements of test case resources. Instead of spending time on developing models from scratch, already developed models of test case resources from other platforms such as SOLIDWORKS, DELMIAV5, etc. should have used. Of course, this approach adds another issue of interfaces which are required to access these models from other platforms, but this could have achieved by asking help from professional users.

Fourth approach: Use of actual physical production resources for test case systems. Although this approach is quite costly, but it can provide more realistic and authentic results. One alternate approach of achieving this is to share the information of research with some small sized manufacturing company and use their redundant resources for the purposes of research.

Fifth approach: Input from real stakeholders. A few interviews of all the stakeholders could have added a more realistic perspective to the proposed implementation.

8. Conclusion and Suggestion

In this thesis, the concept of Resource Description and its executable capabilities is implemented from the perspective of real-world stakeholders. The idea is that the vendors of physical production resources also develop the software packages which provide executable capabilities via exposed interfaces. The consumer of these physical production resources is system integrator. The system integrator develops a high-level control system using the exposed interfaces of vendor provided software packages.

The above-mentioned scenario was implemented in this thesis successfully in a limited scope. The requirements for the objectives were set using the above-mentioned scenario and results were tested by developing test case for the defined scenario. The achievement of the goals relied heavily on the development of software artefacts and most of the time in this thesis was spent on software development processes. In fact, 80 percent of the total time of thesis was spent on development of software artefacts.

As a result, a simulation was created and results were demonstrated using different use cases. The use cases were representation of different kind of reconfiguration scenarios of the real-world manufacturing environment. The results in this research were created and then analysed qualitatively.

In this research, the proposed implementation can automatically execute a specific template of master recipe in a specific platform. The process of creating the master recipe was conducted manually. One valuable area of research would be to automate even the process of creating the master recipe. This would fall under the scope of automatic orchestration.

As suggested in chapter 7, the biggest limitation of this research was that the implementation is performed in only one platform, namely nxtSTUDIO, which is based on IEC 61499. To make the concept of resource description and its executable capability work in real-world, a cross platform implementation is required. The next step in this research would be to implement this concept in a cross platform environment and then analyse its feasibility.

The efficiency and effect of complexity on the proposed implementation would also be an important concern if this concept is to apply in real-world. A real-world test case analysis of the proposed implementation could significantly enrich this concept of resource description and executable capability.

In conclusion, the proposed implementation of the resource description and its executable capability could drastically simplify the reconfiguration process of the manufacturing systems in the control software environment. The concepts require more testing and analysis, but if realized, will add significant value to manufacturing systems.

References

[1] ReCaM-project, “Rapid Reconfiguration of Flexible Production Systems,” ReCaM Consortium, [Online]. Available: http://recam-project.eu/. [Accessed 23 April 2017].

[2] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy and H. Van Brussel, “Reconfigurable Manufacturing Systems,” *CIRP Annals - Manufacturing Technology,* vol. 48, no. 2, pp. 527-540, 1999.

[3] E. Järvenpää, N. Siltala and M. Lanz, “Formal Resource and Capability Descriptions Supporting Rapid Reconfiguration of Assembly Systems,” in *proceedings of the 12th Conference on Automation Science and Engineering, and International Symposium on Assembly and Manufacturing*, 2016.

[4] V. Vyatkin, “IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review,” *IEEE Transactions on Industrial Informatics,* vol. 7, no. 4, pp. 768 - 781, 2011.

[5] A. Zoitl, T. Strasser, C. Sunder and T. Baier, “Is IEC 61499 in harmony with IEC 61131-3?,” *IEEE INDUSTRIAL ELECTRONICS MAGAZINE,* pp. 49-55, 2009.

[6] A. Zoitl and R. Lewis, “Chapter 2: IEC 61499 models and concepts,” in *Modelling Control Systems Using IEC 61499*, London, The Institution of Engineering and Technology, 2014, pp. 21-43.

[7] T. Strasser and A. Zoitl, “Application, System, Device and Resource Model,” in *Distributed Control Applications*, CRC Press, 2016, pp. 17-18.

[8] A. Zoitl and R. Lewis, “Chapter 3: Defining Function Block Types,” in *Modelling Control Systems Using IEC 61499* , London, The Institution of Engineering and Technology, 2014, pp. 45-68.

[9] nxtControl, “nxtSTUDIO,” nxtControl Gmbh, [Online]. Available: http://www.nxtcontrol.com/en/engineering/. [Accessed 23 April 2017].

[10] nxtControl, “nxtLib,” nxtControl Gmbh, [Online]. Available: http://www.nxtcontrol.com/en/libraries/. [Accessed 23 April 2017].

[11] nxtControl, “nxtDCS mini Data sheet,” nxtControl Gmbh, [Online]. Available: http://www.nxtcontrol.com/fileadmin/main/download/nxtDCSmini_DCS10_data_sheet_en.pdf. [Accessed 23 April 2017].

[12] C. R. Kothari, “Research Methodology: An Introduction,” in *Research Methodology: Methods and Techniques*, New Delhi, New Age International, 2004, pp. 1-22.

[13] M. P. Uysal, “Towards a Software Engineering Research Framework: Extending Design Science Research,” *International Research Journal of Engineering and Technology,* vol. 3, no. 2, pp. 22-26, 2016.

[14] Y. Cao, K. Lu, X. Li and Y. Zang, “Accurate Numerical Methods for Computing 2D and 3D Robot Workspace,” *International Journal of Advanced Robotic Systems,* vol. 8, no. 6, pp. 1-13, 2011.

[15] G. Engel, T. Greiner and S. Seifert, “Two-Stage Orchestration Approach for Plug and Produce Based on Semantic Behavior Models,” in *IEEE Tenth International Conference on Semantic Computing*, Laguna Hills, CA, USA, 2016.

[16] M. D. Minicis, F. Giordano, F. Poli and M. M. Schiraldi, “Recipe Development Process Re-Design with ANSI/ISA-88 Batch Control Standard in the Pharmaceutical Industry,” *International Journal of Engineering Business Management,* vol. 6, 2014.

[17] G. Morel, P. Valckenaers, J.-M. Faure, C. E. Pereira and C. Diedrich, “Manufacturing Plant Control Challenges and Issues,” *Control Engineering Practice,* vol. 15, no. 11, pp. 1321-1331, 2007.

[18] K. Thramboulidis, “Challenges in the development of Mechatronic systems: The Mechatronic Component,” in *IEEE International Conference on Emerging technologies and Factory Automation*, Hamburg, 2008.

[19] nxtControl , “nxtDCSmini,” nxtControl Gmbh, [Online]. Available: http://www.nxtcontrol.com/en/control-solution/. [Accessed 20 April 2017].

Appendices

Appendix A.1: Orchestration Process for Test Case Assembly System

Step1: Identify the Sequence of Executable Capabilities which are required to convert the work pieces into final product. Also identify the workpieces or final product for which Executable Capabilities are performed.

Sequence Number	Executable Capability Name	Workpiece/Final Product
1	workpiece	U-shaped workpiece
2	workpiece	Square shaped work piece
3	MoveAbsolute	U-shaped workpiece
4	MoveAbsolute	Square shaped work piece
5	Close	U-shaped workpiece
6	Open	Square shaped work piece
7	Transport	U-shaped workpiece
8	Transport	Square shaped work piece
9	MoveAbsolute	U-shaped workpiece
10	MoveAbsolute	Square shaped work piece
11	InternalGrip	U-shaped workpiece
12	ExternalGrip	Square shaped work piece
13	workpiece	U-shaped workpiece
14	workpiece	U-shaped workpiece
15	workpiece	Square shaped work piece
16	workpiece	Square shaped work piece
17	MoveAbsolute	U-shaped workpiece
18	MoveAbsolute	Square shaped work piece
19	MoveAbsolute	U-shaped workpiece
20	MoveAbsolute	Square shaped work piece
21	MoveAbsolute	U-shaped workpiece
22	Release	U-shaped workpiece
23	workpiece	U-shaped workpiece
24	workpiece	U-shaped workpiece
25	MoveAbsolute	U-shaped workpiece
26	Transport	U-shaped workpiece
27	MoveAbsolute	Square shaped work piece
28	Release	Square shaped work piece
29	workpiece	Final Product
30	workpiece	Final Product
31	MoveAbsolute	Square shaped work piece
32	workpiece	Final Product
33	workpiece	Final Product
34	Transport	Final Product
35	workpiece	Final Product

Step2: For every sequence required to convert the workpieces into final product, identify the order in which the steps of sequence must be performed. This is done by identifying the previous and next of each step in the sequence. Note that if there is no previous or next then it is identified as 0. Also, if there are multiple steps then they are separated by hyphen.

Sequence Number	Previous Sequence	Next Sequence	Executable Capability Name
1	0-	0-	workpiece
2	0-	0-	workpiece
3	0-	7-	MoveAbsolute
4	0-	8-	MoveAbsolute
5	0-	0-	Close
6	0-	0-	Open
7	3-	9-	Transport
8	4-	10-	Transport
9	7-	11-	MoveAbsolute
10	8-	12-	MoveAbsolute
11	9-	13-14-	InternalGrip
12	10-	15-16-	ExternalGrip
13	11-	17-	workpiece
14	11-	17-	workpiece
15	12-	18-	workpiece
16	12-	18-	workpiece
17	13-14-	19-	MoveAbsolute
18	15-16-	20-	MoveAbsolute
19	17-	21-	MoveAbsolute
20	18-	27-	MoveAbsolute
21	19-	22-	MoveAbsolute
22	21-	23-24-	Release
23	22-	25-	workpiece
24	22-	25-	workpiece
25	23-24-	26-	MoveAbsolute
26	25-	27-	Transport
27	20-26-	28-	MoveAbsolute
28	27-	29-30-	Release
29	28-	31-	workpiece
30	28-	31-	workpiece
31	29-30-	32-33-	MoveAbsolute
32	31-	34-	workpiece
33	31-	34-	workpiece
34	32-33-	35-	Transport
35	34-	0-	workpiece

Step 3: Parameters are assigned to each step of the sequence according to the physical layout of the test case assembly system. The required Executable Capabilities are mapped to the resources by assigning the ResourceIDs.

Seq Num	PrevSeq	Next Seq	Resour ceIDs	EC Name	Parameters
1	0-	0-	cnv1	workpiece	wp1=false,wp2=true,wpi=0
2	0-	0-	cnv2	workpiece	wp1=true,wp2=false,wpi=100
3	0-	7-	m2dof1	MoveAbsolute	xDest=45,yDest=100
4	0-	8-	m2dof2	MoveAbsolute	xDest=235,yDest=50
5	0-	0-	grp1	Close	
6	0-	0-	grp2	Open	
7	3-	9-	cnv1	Transport	xPos=true,xNeg=false
8	4-	10-	cnv2	Transport	xPos=false,xNeg=true
9	7-	11-	m2dof1	MoveAbsolute	xDest=45,yDest=185
10	8-	12-	m2dof2	MoveAbsolute	xDest=235,yDest=10
11	9-	13-14-	grp1	InternalGrip	
12	10-	15-16-	grp2	ExternalGrip	
13	11-	17-	cnv1	workpiece	wp1=false,wp2=false,wpi=0
14	11-	17-	grp1	workpiece	wp1=false,wp2=true
15	12-	18-	cnv2	workpiece	wp1=false,wp2=false,wpi=100
16	12-	18-	grp2	workpiece	wp1=true,wp2=false
17	13-14-	19-	m2dof1	MoveAbsolute	xDest=45,yDest=100
18	15-16-	20-	m2dof2	MoveAbsolute	xDest=235,yDest=50
19	17-	21-	m2dof1	MoveAbsolute	xDest=255,yDest=100
20	18-	27-	m2dof2	MoveAbsolute	xDest=45,yDest=50
21	19-	22-	m2dof1	MoveAbsolute	xDest=255,yDest=185
22	21-	23-24-	grp1	Release	
23	22-	25-	cnv3	workpiece	wp1=false,wp2=true,wpi=0
24	22-	25-	grp1	workpiece	wp1=false,wp2=false
25	23-24-	26-	m2dof1	MoveAbsolute	xDest=255,yDest=100
26	25-	27-	cnv3	Transport	xPos=true,xNeg=false
27	20-26-	28-	m2dof2	MoveAbsolute	xDest=45,yDest=170
28	27-	29-30-	grp2	Release	
29	28-	31-	cnv3	workpiece	wp1=true,wp2=true,wpi=100
30	28-	31-	grp2	workpiece	wp1=false,wp2=false
31	29-30-	32-33-	m2dof2	MoveAbsolute	xDest=45,yDest=100
32	31-	34-	cnv3	workpiece	wp1=false,wp2=false,wpi=0
33	31-	34-	cnv4	workpiece	wp1=true,wp2=true,wpi=0
34	32-33-	35-	cnv4	Transport	xPos=true,xNeg=false
35	34-	0-	cnv4	workpiece	wp1=false,wp2=false,wpi=0

Step 4: Master Recipe for Test Case Assembly System

The output of the Orchestration process is converted to the Master Recipe by following the proposed template.

<MasterRecipe>
<Sequence Num="1">Prev=0-,Next=0-,ResourceID=cnv1,EC=workpiece,wp1=false,wp2=true,wpi=0</Sequence>
<Sequence Num="2">Prev=0-,Next=0-,ResourceID=cnv2,EC=workpiece,wp1=true,wp2=false,wpi=100</Sequence>
<Sequence Num="3">Prev=0-,Next=7-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=45,yDest=100</Sequence>
<Sequence Num="4">Prev=0-,Next=8-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=235,yDest=50</Sequence>
<Sequence Num="5">Prev=0-,Next=0-,ResourceID=grp1,EC=Close</Sequence>
<Sequence Num="6">Prev=0-,Next=0-,ResourceID=grp2,EC=Open</Sequence>
<Sequence Num="7">Prev=3-,Next=9-,ResourceID=cnv1,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="8">Prev=4-,Next=10-,ResourceID=cnv2,EC=Transport,xPos=false,xNeg=true</Sequence>
<Sequence Num="9">Prev=7-,Next=11-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=45,yDest=185</Sequence>
<Sequence Num="10">Prev=8-,Next=12-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=235,yDest=105</Sequence>
<Sequence Num="11">Prev=9-,Next=13-14-,ResourceID=grp1,EC=InternalGrip</Sequence>
<Sequence Num="12">Prev=10-,Next=15-16-,ResourceID=grp2,EC=ExternalGrip</Sequence>
<Sequence Num="13">Prev=11-,Next=17-,ResourceID=cnv1,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>
<Sequence Num="14">Prev=11-,Next=17-ResourceID=grp1,EC=workpiece,wp1=false,wp2=true</Sequence>
<Sequence Num="15">Prev=12-,Next=18-,ResourceID=cnv2,EC=workpiece,wp1=false,wp2=false,wpi=100</Sequence>
<Sequence Num="16">Prev=12-,Next=18-,ResourceID=grp2,EC=workpiece,wp1=true,wp2=false</Sequence>
<Sequence Num="17">Prev=13-14-,Next=19-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=45,yDest=100</Sequence>
<Sequence Num="18">Prev=15-16-,Next=20-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=235,yDest=50</Sequence>
<Sequence Num="19">Prev=17-,Next=21-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=255,yDest=100</Sequence>
<Sequence Num="20">Prev=18-,Next=27-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=45,yDest=50</Sequence>
<Sequence Num="21">Prev=19-,Next=22-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=255,yDest=185</Sequence>
<Sequence Num="22">Prev=21-,Next=23-24-,ResourceID=grp1,EC=Release</Sequence>
<Sequence Num="23">Prev=22-,Next=25-,ResourceID=cnv3,EC=workpiece,wp1=false,wp2=true,wpi=0</Sequence>

<Sequence Num="24">Prev=22-,Next=25-,ResourceID=grp1,EC=workpiece,wp1=false,wp2=false</Sequence>
<Sequence Num="25">Prev=23-24-,Next=26-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=255,yDest=100</Sequence>
<Sequence Num="26">Prev=25-,Next=27-,ResourceID=cnv3,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="27">Prev=20-26-,Next=28-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=45,yDest=170</Sequence>
<Sequence Num="28">Prev=27-,Next=29-30-,ResourceID=grp2,EC=Release</Sequence>
<Sequence Num="29">Prev=28-,Next=31-,ResourceID=cnv3,EC=workpiece,wp1=true,wp2=true,wpi=100</Sequence>
<Sequence Num="30">Prev=28-,Next=31-,ResourceID=grp2,EC=workpiece,wp1=false,wp2=false</Sequence>
<Sequence Num="31">Prev=29-30-,Next=32-33-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=45,yDest=100</Sequence>
<Sequence Num="32">Prev=31-,Next=34-,ResourceID=cnv3,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>
<Sequence Num="33">Prev=31-,Next=34-,ResourceID=cnv4,EC=workpiece,wp1=true,wp2=true,wpi=0</Sequence>
<Sequence Num="34">Prev=32-33-,Next=35-,ResourceID=cnv4,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="35">Prev=34-,Next=0-,ResourceID=cnv4,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>
</MasterRecipe>

Appendix A.2: Orchestration Process for Reconfigured Assembly System

Step1: Identify the Sequence of Executable Capabilities which are required to convert the work pieces into final product. Also identify the workpieces or final product for which Executable Capabilities are performed.

Sequence Number	Executable Capability Name	Workpiece/Final Product
1	workpiece	U-shaped workpiece 1
2	workpiece	Square shaped work piece 1
3	MoveAbsolute	Square shaped work piece 1
4	Open	Square shaped work piece 1
5	Transport	U-shaped workpiece 1
6	Transport	Square shaped work piece 1
7	MoveAbsolute	Square shaped work piece 1
8	ExternalGrip	Square shaped work piece 1
9	workpiece	Square shaped work piece 1
10	workpiece	Square shaped work piece 1
11	MoveAbsolute	Square shaped work piece 1
12	MoveAbsolute	Square shaped work piece 1
13	MoveAbsolute	Square shaped work piece 1
14	Release	Square shaped work piece 1
15	workpiece	Final Product 1
16	workpiece	Square shaped work piece 1
17	MoveAbsolute	Square shaped work piece 1
18	workpiece	Final Product 1
19	workpiece	Final Product 1
20	Transport	Final Product 1
21	workpiece	Final Product 1
22	workpiece	U-shaped workpiece 2
23	workpiece	Square shaped work piece 2
24	MoveAbsolute	Square shaped work piece 2
25	Open	Square shaped work piece 2
26	Transport	U-shaped workpiece 2
27	Transport	Square shaped work piece 2
28	MoveAbsolute	Square shaped work piece 2
29	ExternalGrip	Square shaped work piece 2
30	workpiece	Square shaped work piece 2
31	workpiece	Square shaped work piece 2
32	MoveAbsolute	Square shaped work piece 2
33	MoveAbsolute	Square shaped work piece 2
34	MoveAbsolute	Square shaped work piece 2
35	Release	Square shaped work piece 2
36	workpiece	Final Product 2
37	workpiece	Square shaped work piece 2
38	MoveAbsolute	Square shaped work piece 2
39	workpiece	Final Product 2

40	workpiece	Final Product 2
41	Transport	Final Product 2
42	workpiece	Final Product 2

Step2: For every step in the sequence required to convert the workpieces into final product, identify the order in which the steps must be performed. This is done by identifying the previous and next of each step in the sequence. Note that if there is no previous or next then it is identified as 0.

Sequence Number	Prev Seq	Next Seq	EC Name
1	0-	5-	workpiece
2	0-	6-	workpiece
3	0-	7-	MoveAbsolute
4	0-	0-	Open
5	1-	13-	Transport
6	2-	7-	Transport
7	3-6-	8-	MoveAbsolute
8	7-	9-10-	ExternalGrip
9	8-	11-	workpiece
10	8-	11-	workpiece
11	9-10-	12-	MoveAbsolute
12	11-	13-	MoveAbsolute
13	5-12-	14-	MoveAbsolute
14	13-	15-16-	Release
15	14-	17-	workpiece
16	14-	17-	workpiece
17	15-16-	18-19-	MoveAbsolute
18	17-	20-	workpiece
19	17-	20-	workpiece
20	18-19-	21-	Transport
21	20-	0-	workpiece
22	0-	26-	workpiece
23	0-	27-	workpiece
24	0-	28-	MoveAbsolute
25	0-	0-	Open
26	22-	34-	Transport
27	23-	28-	Transport
28	24-27-	29-	MoveAbsolute
29	28-	30-31-	ExternalGrip
30	29-	32-	workpiece
31	29-	32-	workpiece
32	30-31-	33-	MoveAbsolute
33	32-	34-	MoveAbsolute
34	26-33-	35-	MoveAbsolute
35	34-	36-37-	Release
36	35-	38-	workpiece
37	35-	38-	workpiece
38	36-37-	39-40-	MoveAbsolute
39	38-	41-	workpiece

40	38-	41-	workpiece
41	39-40-	42-	Transport
42	41-	0-	workpiece

Step3: Parameters are assigned to each step of the sequence according to the physical layout of the reconfigured assembly system. The required Executable Capabilities are mapped to the Resources by assigning the ResourceIDs.

Seq	Prev Seq	Next Seq	ResIDs	EC Name	Parameters
1	0-	5-	cnv1	workpiece	wp1=false,wp2=true,wpi=0
2	0-	6-	cnv3	workpiece	wp1=true,wp2=false,wpi=100
3	0-	7-	m2dof1	MoveAbsolute	xDest=235,yDest=50
4	0-	0-	grp1	Open	
5	1-	13-	cnv1	Transport	xPos=true,xNeg=false
6	2-	7-	cnv3	Transport	xPos=false,xNeg=true
7	3-6-	8-	m2dof1	MoveAbsolute	xDest=235,yDest=105
8	7-	9-10-	grp1	ExternalGrip	
9	8-	11-	cnv3	workpiece	wp1=false,wp2=false,wpi=100
10	8-	11-	grp1	workpiece	wp1=true,wp2=false
11	9-10-	12-	m2dof1	MoveAbsolute	xDest=235,yDest=50
12	11-	13-	m2dof1	MoveAbsolute	xDest=45,yDest=50
13	5-12-	14-	m2dof1	MoveAbsolute	xDest=45,yDest=170
14	13-	15-16-	grp1	Release	
15	14-	17-	cnv1	workpiece	wp1=true,wp2=true,wpi=100
16	14-	17-	grp1	workpiece	wp1=false,wp2=false
17	15-16-	18-19-	m2dof1	MoveAbsolute	xDest=45,yDest=100
18	17-	20-	cnv1	workpiece	wp1=false,wp2=false,wpi=0
19	17-	20-	cnv2	workpiece	wp1=true,wp2=true,wpi=0
20	18-19-	21-	cnv2	Transport	xPos=true,xNeg=false
21	20-	0-	cnv2	workpiece	wp1=false,wp2=false,wpi=0
22	0-	26-	cnv4	workpiece	Parameters
23	0-	27-	cnv6	workpiece	wp1=false,wp2=true,wpi=0
24	0-	28-	m2dof2	MoveAbsolute	wp1=true,wp2=false,wpi=100
25	0-	0-	grp2	Open	xDest=235,yDest=50
26	22-	34-	cnv4	Transport	
27	23-	28-	cnv6	Transport	xPos=true,xNeg=false
28	24-27-	29-	m2dof2	MoveAbsolute	xPos=false,xNeg=true
29	28-	30-31-	grp2	ExternalGrip	xDest=235,yDest=105
30	29-	32-	cnv6	workpiece	
31	29-	32-	grp2	workpiece	wp1=false,wp2=false,wpi=100
32	30-31-	33-	m2dof2	MoveAbsolute	wp1=true,wp2=false
33	32-	34-	m2dof2	MoveAbsolute	xDest=235,yDest=50
34	26-33-	35-	m2dof2	MoveAbsolute	xDest=45,yDest=50
35	34-	36-37-	grp2	Release	xDest=45,yDest=170
36	35-	38-	cnv4	workpiece	
37	35-	38-	grp2	workpiece	wp1=true,wp2=true,wpi=100
38	36-37-	39-40-	m2dof2	MoveAbsolute	wp1=false,wp2=false
39	38-	41-	cnv4	workpiece	xDest=45,yDest=100
40	38-	41-	cnv5	workpiece	wp1=false,wp2=false,wpi=0

41	39-40-	42-	cnv5	Transport	wp1=true,wp2=true,wpi=0
42	41-	0-	cnv5	workpiece	xPos=true,xNeg=false

Step4: Master Recipe

The output of the Orchestration process is converted to the Master Recipe according to the proposed template.

<MasterRecipe>
<Sequence Num="1">Prev=0-,Next=5-,ResourceID=cnv1,EC=workpiece,wp1=false,wp2=true,wpi=0</Sequence>
<Sequence Num="2">Prev=0-,Next=6-,ResourceID=cnv3,EC=workpiece,wp1=true,wp2=false,wpi=100</Sequence>
<Sequence Num="3">Prev=0-,Next=7-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=235,yDest=50</Sequence>
<Sequence Num="4">Prev=0-,Next=0-,ResourceID=grp1,EC=Open</Sequence>
<Sequence Num="5">Prev=1-,Next=13-,ResourceID=cnv1,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="6">Prev=2-,Next=7-,ResourceID=cnv3,EC=Transport,xPos=false,xNeg=true</Sequence>
<Sequence Num="7">Prev=3-6-,Next=8-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=235,yDest=105</Sequence>
<Sequence Num="8">Prev=7-,Next=9-10-,ResourceID=grp1,EC=ExternalGrip</Sequence>
<Sequence Num="9">Prev=8-,Next=11-,ResourceID=cnv3,EC=workpiece,wp1=false,wp2=false,wpi=100</Sequence>
<Sequence Num="10">Prev=8-,Next=11-,ResourceID=grp1,EC=workpiece,wp1=true,wp2=false</Sequence>
<Sequence Num="11">Prev=9-10-,Next=12-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=235,yDest=50</Sequence>
<Sequence Num="12">Prev=11-,Next=13-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=45,yDest=50</Sequence>
<Sequence Num="13">Prev=5-12-,Next=14-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=45,yDest=170</Sequence>
<Sequence Num="14">Prev=13-,Next=15-16-,ResourceID=grp1,EC=Release</Sequence>
<Sequence Num="15">Prev=14-,Next=17-,ResourceID=cnv1,EC=workpiece,wp1=true,wp2=true,wpi=100</Sequence>
<Sequence Num="16">Prev=14-,Next=17-,ResourceID=grp1,EC=workpiece,wp1=false,wp2=false</Sequence>
<Sequence Num="17">Prev=15-16-,Next=18-19-,ResourceID=m2dof1,EC=MoveAbsolute,xDest=45,yDest=100</Sequence>
<Sequence Num="18">Prev=17-,Next=20-,ResourceID=cnv1,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>
<Sequence Num="19">Prev=17-,Next=20-,ResourceID=cnv2,EC=workpiece,wp1=true,wp2=true,wpi=0</Sequence>
<Sequence Num="20">Prev=18-19-,Next=21-,ResourceID=cnv2,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="21">Prev=20-,Next=0-,ResourceID=cnv2,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>

<Sequence Num="22">Prev=0-,Next=26-,ResourceID=cnv4,EC=workpiece,wp1=false,wp2=true,wpi=0</Sequence>
<Sequence Num="23">Prev=0-,Next=27-,ResourceID=cnv6,EC=workpiece,wp1=true,wp2=false,wpi=100</Sequence>
<Sequence Num="24">Prev=0-,Next=28-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=235,yDest=50</Sequence>
<Sequence Num="25">Prev=0-,Next=0-,ResourceID=grp1,EC=Open</Sequence>
<Sequence Num="26">Prev=22-,Next=34-,ResourceID=cnv4,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="27">Prev=23-,Next=28-,ResourceID=cnv6,EC=Transport,xPos=false,xNeg=true</Sequence>
<Sequence Num="28">Prev=24-27-,Next=29-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=235,yDest=105</Sequence>
<Sequence Num="29">Prev=28-,Next=30-31-,ResourceID=grp2,EC=ExternalGrip</Sequence>
<Sequence Num="30">Prev=29-,Next=32-,ResourceID=cnv6,EC=workpiece,wp1=false,wp2=false,wpi=100</Sequence>
<Sequence Num="31">Prev=29-,Next=32-,ResourceID=grp2,EC=workpiece,wp1=true,wp2=false</Sequence>
<Sequence Num="32">Prev=30-31-,Next=33-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=235,yDest=50</Sequence>
<Sequence Num="33">Prev=32-,Next=34-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=45,yDest=50</Sequence>
<Sequence Num="34">Prev=26-33-,Next=35-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=45,yDest=170</Sequence>
<Sequence Num="35">Prev=34-,Next=36-37-,ResourceID=grp2,EC=Release</Sequence>
<Sequence Num="36">Prev=35-,Next=38-,ResourceID=cnv4,EC=workpiece,wp1=true,wp2=true,wpi=100</Sequence>
<Sequence Num="37">Prev=35-,Next=38-,ResourceID=grp2,EC=workpiece,wp1=false,wp2=false</Sequence>
<Sequence Num="38">Prev=36-37-,Next=39-40-,ResourceID=m2dof2,EC=MoveAbsolute,xDest=45,yDest=100</Sequence>
<Sequence Num="39">Prev=38-,Next=41-,ResourceID=cnv4,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>
<Sequence Num="40">Prev=38-,Next=41-,ResourceID=cnv5,EC=workpiece,wp1=true,wp2=true,wpi=0</Sequence>
<Sequence Num="41">Prev=39-40-,Next=42-,ResourceID=cnv5,EC=Transport,xPos=true,xNeg=false</Sequence>
<Sequence Num="42">Prev=41-,Next=0-,ResourceID=cnv5,EC=workpiece,wp1=false,wp2=false,wpi=0</Sequence>
</MasterRecipe>